December 1983

Report No. STAN-B-83-989

A Programming and Problem-Solving Seminar

by

Donald E. Knuth and Joseph S. Weening

Department of Computer Science

Stanford University
Stanford, CA 94305

A PROGRAMMING AND PROBLEM-SOLVING SEMINAR

by

Donald E. Knuth and Joseph S. Weening

This report contains edited transcripts of the discussions held in Stanford's course CS 204, Problem Seminar, during autumn quarter **1981.** Since the topics span a large range of ideas in computer science, and since most of the important research paradigms and programming paradigms were touched on during the discussions, these notes may be of interest to graduate students of computer science at other universities, as well as to their professors and to professional people in the "real world."

The present report is the fourth in a series of such transcripts, continuing the tradition established in CS606 (Michael J. Clancy, 1977), CS707 (Chris Van Wyk, 1979), CS863 (Allan A. Miller, **1981).**

The production of this report was partially supported by a National Science Foundation fellowship and by NSF grant MCS-8300984.

Table of Contents

Introduction 1
October 1 2

Problem 1: Penny Flipping 3
October 6 4
October 8 8
October 13 11
Solutions 14

Problem 2: Archival Files 17
October 15 17
October 20 21
October 22 24
October 27 26
Solutions 27

Problem 3: 2 x 2 x 2 Rubicity 30
October 27 31
October 29 33
November 3 35
November 5 35
November 10 37
Solutions 39

Problem 4: Fractal Topography 41
November 10 41
November 12 42
November 17 47
November 19 50
November 24 53
Solutions 55

Problem 5: Question answering 63
December 1 63
December 3 66
December 8 70
Solutions 90

Introduction

The work in CS 204 consisted of five problems. Each one was presented and discussed during four class sessions, over a period of two weeks. As ideas were developed, members of the class, working in groups, implemented their solutions and reported their results to the class.

Each problem is presented here, followed by the class discussions, and is concluded with a summary of the results obtained. In the interest of brevity, the participants are identified by their initials, which correspond to the names in the following list.

DEK	Donald E. Knuth (instructor)
JSW	Joseph S. Weening (teaching assistant)

RJA	Richard J. Anderson
KPB	Kenneth P. Brooks
DMC	David M. Chelberg
KC	Kelly Cor the11
JF	Joan Feigenbaum
DEF	David E. Foulser
AH	Armin Haken
DPH	David P. Helmbold
LAH	Lane A. Hemachandra
JEH	John E. Hershberger
ARK	Anna R. Karlin
SSK	Sanjay Kasturia
YM	Yoni Malachi
TPM	Timothy P. Mann
JLM	James L. McDonald
RJM	Robert J. Meier
JCM	Jeffrey C. Mogul
YOM	Yoram 0. Moses
RGS	Ralph G. Saavedra
AIS	Andrew I. Shore
ARS	Alan R. Siegel
AVG	Allen Van Gelder

Class notes for Thursday, October 1

At the first class meeting, DEK handed out all five problem descriptions, and made some introductory remarks about the purpose of the course and the work expected from the class. He emphasized that the object of the course was to work together in finding solutions, and that competition was to be avoided. Three of the problems have no published solution, and the fourth and fifth also have potential for new ideas in their solutions.

Because of this, trial-and-error approaches to problem-solving are encouraged, and the reports handed in should include a description of methods that looked promising but failed. They should also include some introspection, describing the way in which various ideas were thought of and which ideas proved useful. This comes closest to methods used to solve problems in all areas of computer science research.

DEK had brought along a copy of the course evaluation from a previous CS 204 class. Comments by the students in that class stressed the heavy workload, and generally indicated that the course was a useful way to learn problem-solving skills, though not in a formal way.

Thursday, October 1

Problem 1: Penny Flipping

A stack of n identical pennies has been neatly arranged so that each coin is "heads up." Then the following operations are done, ad infinitum:

(1) Flip the top coin.
(2) Flip the whole stack.
(3) Flip the top two coins.
(4) Flip the whole stack.
(5) Flip the top three coins.
(6) Flip the whole stack.
...
$(2n-3)$ Flip the top $n-1$ coins.
$(2n-2)$ Flip the whole stack.
$(2n-1)$ Start over again at (1).

For example, when $n = 4$ the stack goes through the following configurations of heads and tails (reading downwards):

Starting position	HHHH					
(1) Flip 1 coin	THHH	TTHT	TTTH	HTTT	HHTH	HHHT
(2) Flip 4 coins	TTTH	HTHH	THHH	HHHT	THTT	HTTT
(3) Flip 2 coins	HHTH	HTHH	THHH	TTHT	THTT	HTTT
(4) Flip 4 coins	THTT	TTHT	TTTH	HTHH	HHTH	HHHT
(5) Flip 3 coins	HTHT	THHT	HHHH	THTH	HTTH	TTTT
(6) Flip 4 coins	HTHT	HTTH	TTTT	THTH	THHT	HHHH

after which the pattern repeats in a cycle of length 36.

Question: If we do one flip per unit time, at what times are all the heads up? The answer to this question, when $n = 4$, is that HHHH occurs at time $36m$ and $36m + 17$, for $m = 0, 1, 2, \ldots$, and at no other times.

A bit of study reveals that the stack will indeed return to its initial state infinitely often, regardless of the value of n. Furthermore if this happens at times $0, t_1, t_2, \ldots$, there will be a first time $t_k > 0$ such that t_k is a multiple of $2n - 2$; this means that all heads will be up at time t_k and that step (1) of the algorithm will be next, so the process will repeat itself. Then the heads-up times will be precisely $t_k m$, $t_k m + t_1, \ldots, t_k m + t_{k-1}$, for $m \geq 0$.

This problem has been used as an exercise in programming texts for several years (e.g., in SIMULA begin by Birtwistle, Dahl, Myhrhaug, and Nyganrd, 1973), because it makes a nice illustration of elementary concepts. But the methods proposed for solution in these texts require time proportional to t_k, so they are unsuitable unless n is very small. In fact, some years went by before the case $n = 54$ was solved; and as recently as 1981 a computer science professor in southern California decided to extend the known results by coding the problem in Apple II assembly language, but he turned his machine off after it had run more than 24 hours on the case $n = 113$.

Your task is to find a way to solve this problem for $1 \le n \le 120$, using only a few seconds of computer time.

Class notes for Tuesday, October 6

DEK started the class by explaining that Problem 1 is really a "toy" problem, but although it does not have any real-world applications, working on it is a good way to learn problem-solving techniques.

A simpler case, namely flipping the top coin, then the top two, etc., and flipping the whole stack of coins only once per iteration, has been considered before, and a solution was presented in *Mathematics Magazine* in March 1981 (pp. 51-59). The solution to that problem is a sequence that has a much more regular appearance than the sequence in our version.

GSK said that he had tried two approaches to the problem. The first was to try to find a recurrence relation that would give a description of each state (i.e., sequence of H's and T's) from the previous state and the current operation being performed. For example, if we are flipping the top k coins, and x_i represents the state of the ith coin, then

$$\begin{aligned} x'_i &= \bar{x}_{k+1-i}, && \text{if } i \le k \\ x'_i &= x_i, && \text{if } i > k \end{aligned}$$

describes the transformation, where x'_i means the new value of x_i, and the "bar" operator means to switch from H to T or vice versa. DEK mentioned that this resembles methods that others have used in trying to solve the problem, none of which have been very successful.

The other approach is to follow each coin as it moves through the stack. In the case $n = 4$, we would now represent the transitions as

Starting position	1234					
(1) Flip 1 coin	$\bar{1}234$	$\bar{3}\bar{1}2\bar{4}$	$\bar{2}\bar{3}\bar{1}4$	$\bar{1}234$	$\bar{3}\bar{1}2\bar{4}$	$231\bar{4}$
(2) Flip 4 coins	4321	4213	4132	4321	4213	4132
(3) Flip 2 coins	3421	$2\bar{4}13$	$\bar{1}\bar{4}\bar{3}2$	$\bar{3}\bar{4}2\bar{1}$	$\bar{2}4\bar{1}\bar{3}$	$\bar{1}\bar{4}\bar{3}2$
(4) Flip 4 coins	$\bar{1}2\bar{4}\bar{3}$	$\bar{3}\bar{1}4\bar{2}$	2341	1243	$31\bar{4}2$	$\bar{2}\bar{3}\bar{4}\bar{1}$
(5) Flip 3 coins	4213	4132	$\bar{4}\bar{3}\bar{2}\bar{1}$	$\bar{4}2\bar{1}3$	$4\bar{1}\bar{3}2$	4321
(6) Flip 4 coins	$3\bar{1}2\bar{4}$	2%4	1234	3124	2314	1234

The effect of the first $2n - 2$ flips (the first column) is to move 1234 into $3\bar{1}2\bar{4}$, and we can use this to compute the configurations at the bottom of subsequent columns more quickly than by simulating all $2n - 2$ flips in each column. This is because the operations done in each column are now seen to be a permutation that takes any configuration $x_1x_2x_3x_4$ into $x_3\bar{x}_1x_2\bar{x}_4$. Applying this to $3\bar{1}2\bar{4}$ gives $2\bar{3}\bar{1}4$, and so on.

By saving information that at first appears unnecessary, we have reduced the amount of work to be done by a factor proportional to n. This also illustrates that working out a small case by hand can lead to insights that otherwise might not occur; RJM pointed out that in this case it led to an algorithm whose running time is at most $O(n^3)$. DEK also noted that testing a small case can point out what isn't true about a problem; counterexamples are as important as theorems when we are exploring new territory. (For example, we might conjecture that the coin at the bottom of the stack is always still there after 272 − 2 flips, but trying $n = 5$ would disprove this.)

KPB asked, however, whether anyone knew what the complexity of the "brute force" solution really is; this must be known to decide whether any potential algorithm for the solution really saves time. By "brute force" we mean applying the flips successively until we reach a configuration in which all of the coins are heads up. JF noted that finding an all-tails configuration at the bottom of a column was also sufficient; if this occurred in t steps, then all heads would appear after $2t$ steps. Assuming that a flip of k coins requires computer time proportional to k (i.e., that there is no special hardware), and assuming that the first heads-up configuration occurs in column m of a diagram such as shown above, AVG stated that the time to find that configuration is $O(mn^2)$, ***since*** the time needed to do the flips in one column is

$$1 + n + 2 + n + \dots + (n-1) + n = \tfrac{3}{2}n(n-1).$$

Therefore, unless m is large, we may not be able to improve the running time significantly.

DEK told the class, however, that his experiments showed m to be quite large compared to n; thus improvements on the brute force method are worthwhile. It is fairly easy to see that m is bounded by 2^n, so the brute force method takes at most exponential time. This led into a slight digression on the running time of algorithms in general. An exponential time is usually considered far inferior to a polynomial time; for example 2” is worse than n^3 for $n \geq 10$. Thus problems whose only known solutions take exponential time are often considered intractable. However, a polynomial with a large exponent, say n^{1000}, is really far worse than 2” since n^{1000} is infeasible even when $n = 2$; so occasionally an exponential algorithm may be preferred. If there is a polynomial-time algorithm for a problem, however, improvements to it often lead to a reduction in the exponent.

RJM suggested the following way to approach the coin-flipping problem: let the $2n - 2$ flips in one column be represented by a matrix; so for the case $n = 4$ we would have

$$M = \begin{pmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}, \quad \text{so that } M \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 3 \\ -1 \\ 2 \\ -4 \end{pmatrix}.$$

Then the powers of the matrix M, such as

$$M^2 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{and } M^3 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix},$$

describe the transformations that result from multiple columns of flips. By computing the eigenvalues of M, we can find the the smallest k such that M^k is the identity matrix. ARS pointed out that this is too strong a condition; all that we really need is a power of M with no -1's in it.

DEK mentioned that the sequence (flip n, flip k, flip n), which occurs in the statement of the problem, is equivalent to flipping the bottom k coins in the stack. This can be used to speed up manual computations.

DEK now returned to using the notation of permutations to represent the result of flipping coins, and explained how permutations can be written as products of cycles. Given any permutation, say

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 1 & 4 & 7 & 6 & 5 & 2 & 9 & 8 \end{pmatrix}$$

which maps 1 to 3, 2 to 1, etc., we can rewrite it in the form

$$(1\ 3\ 4\ 7\ 2)(5\ 6)(8\ 9),$$

which is a short way of saying that 1, 3, 4, 7, and 2 are a cycle, each one being mapped to the one listed to its right, except that 2 is mapped into 1; 5 and 6 are interchanged; and 8 and 9 are interchanged. Any permutation on a finite set has such a representation. (See section 1.3.3 in volume 1 of *The Art of Computer Programming* for a more detailed explanation.) The cycle notation is especially useful for taking powers of permutations, since the power operation can be done on each cycle independently of the others. For example, the square of the above permutation is (1 4 2 3 7)(5)(G)(8)(9), the fifth power is (1)(2)(3)(4)(5 6)(7)(8 9), and the tenth power is the identity permutation. DEK asked the class how, given the lengths of the cycles of a permutation, to find the smallest power of that permutation which is the identity, and RJA answered that taking the least common multiple of those lengths provided the minimum.

KPB noted that reaching the identity permutation is not necessary here, since we are only interested in whether the coins are heads up or tails. Also, the standard notation for permutations only refers to the positions of the coins, so DEK decided to generalize the notation. One way to do this is to use a. permutation on 2n objects, with the head and tail of each coin being a separate object. Then flipping the first coin in a pile of four would be $\binom{1\ \bar1\ 2\ \bar2\ 3\ \bar3\ 4\ \bar4}{\bar1\ 1\ 2\ \bar2\ 3\ \bar3\ 4\ \bar4}$, and the first column of $2n$ -- 2 flips would result in the permutation $\binom{1\ \bar1\ 2\ \bar2\ 3\ \bar3\ 4\ \bar4}{\bar2\ 2\ 3\ \bar3\ \bar1\ 1\ \bar4\ 4}$, which is $(1\ \bar2\ \bar3\ \bar1\ 2\ 3)(4\ \bar4)$ in cycle notation. This is interpreted as moving the object in position 1 to position $\bar2$, and so on. For n = 5, we get $(1\ \bar4\ \bar2\ \bar5\ \bar3)(\bar1\ 4\ 2\ 5\ 3)$ after 8 flips, and here we see some redundancy in the notation: when there is a cycle in which no coin appears twice (i.e., we don't have both x and $\bar{x}$), then the "complement" of that cycle also appears in the permutation.

If p is the permutation representing the operations done in the first column of the problem diagram, then powers of p will represent subsequent columns. For example, if

$p = (1\ 2\ \bar{3}\ 4\ 5\ \bar{6})$ then $p^3 = (1\ 4)(2\ 5)(3\ 6)$. This corresponds to the matrix

$$M^3 = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

Here we see a potential problem: the absence of **-1's** means that all coins are heads up, but M^3 is not the identity matrix. SSK ted that the matrix equation

$$M^t \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

expresses exactly the fact that there is a $+1$ in each column, which is what we want. Using techniques of block diagonalization, we may be able to solve this equation efficiently.

JF said that block diagonalization is really the same as using the cycle decomposition of the permutations, and DEK explained that in most problems of this nature the permutations contain all the necessary information, and require less space and running time than the matrices.

AVG asked whether the object was just to find t_k, or also to compute the values $t_1, \ldots, t_{k-1}$ that would name all of the heads-up configurations. DEK's answer was that he wanted the class to find the values $t_1, \ldots, t_{k-1}$, and he sketched how we might start to do this. If the permutations $p_1, p_2, \ldots, p_{2n-2}$ are defined to be the operations done by the individual flips in each iteration, then $p = p_1 p_2 \ldots p_{2n-2}$ is the permutation for a whole column, and whenever $p^k p_1 \ldots p_j$ is an "all heads" permutation, we have one of these "unlucky solutions." If, for each j between **1** and 2n -- **1,** we find the smallest such ***k,*** then we will have all of the solutions. KPB told the class that he had enumerated all the cases up to n = 14, and found that the number of such solutions seemed to approach 0 as n increased. DEK said, however, that an examination of more values will show a surprising pattern; he predicts that somebody will discover infinitely many "unlucky solutions."

To end the class, DEK gave an outline of the algorithm that had now been developed: **(1)** compute the permutation p for a whole column of the diagram (the first $2n - 2$ flips); (2) decompose this permutation into cycles, and find the smallest power of each cycle that is all "heads up;" (3) take the least common multiple (lcm) of these numbers to find t_k.

To do step (2), suppose we have a cycle such as $(a\ b\ \bar{c}\ d\ e\ \bar{f})$. The identities of the objects being permuted don't matter, so we can just think of this as a string of heads and tails, i.e., **001001,** where 0 represents heads and 1 represents tails. Then the smallest amount that we can shift this string (with wraparound) until it equals itself is the desired power of the cycle. (In this case it is 3.)

The lcm operation in step (3) has the potential of generating a large answer with a small amount of computation, even though the numbers used are small. This is where the time savings of the algorithm becomes apparent.

Someone raised the question of how large this lcm can be, so that we have a better upper bound on m than the obvious 2^n. This is a famous problem in mathematics called Landau's problem. Let g(n) be the maximum lcm of positive integers that sum to n. It can be shown that

$$\log g(n)\ \sqrt{n\log n}$$

(see J. L. Nicolas, "Order Maximal d'un Element du Groupe S_n des Permutations et 'Highly Composite Numbers' ", *Bull. Soc.* Math. France, 97, 1969, pp. 129–191). Recent work has provided more exact bounds on g(n).

Class notes for Thursday, October 8

The class started by determining how to find efficiently the lcm of a set of integers, which is step (3) of the algorithm developed last time. DMC said that computing the factorization of each integer and applying the definition of the lcm was one way, but DEK said that there was a better way. The relation $\mathrm{lcm}(x,y)\cdot\gcd(x,y) = x \cdot y$ was suggested by LAH, since this reduces the problem to finding the greatest common divisor.

DEK pointed out that this relation resembles several others in mathematics, such as

$$|A\cup B| + |A\cap B| = |A| + |B|$$

and

$$\max(x, y) + \min(x, y) = x + y,$$

where the first holds for finite sets and the second for real numbers. In fact, the relation between the gcd and lcm can be proved from the second equation above, since if

$$x = 2^{x_2}3^{x_3}5^{x_5}\ldots$$
$$Y = 2^{y_2}3^{y_3}5^{y_5}\ldots,$$

then

$$\gcd(x,y) = 2^{\min(x_2,y_2)}3^{\min(x_3,y_3)}5^{\min(x_5,y_5)}\ldots,$$
$$\mathrm{lcm}(x,y) = 2^{\max(x_2,y_2)}3^{\max(x_3,y_3)}5^{\max(x_5,y_5)}\ldots.$$

An efficient algorithm for finding the gcd is, of course, Euclid's algorithm, which runs in about the same time as division of integers. Factoring, although it has never been proved any less efficient, takes so long that nobody today knows a good way to factor 100-digit numbers with less than several years of time on the fastest computers. DEK said that he has never seen an efficient algorithm to compute $\mathrm{lcm}(x, y)$ that did not involve computing $\gcd(x, y)$ also.

DEK asked whether anyone had made progress in finding the "unlucky solutions" to the coin-flipping problem, and RJM started to outline the method that his group was using. If we let $q_k = p_1p_2\ldots p_k$, and let $p = q_{2n-2} = p_1p_2\ldots p_{2n-2}$, where p_i is the permutation corresponding to operation i, then q_k corresponds to doing the first k operations in a column. The idea is to compute each q_k once, and find the patterns of heads and tails that q_k will transform into all heads. If p^j has this pattern for some j, then $p^j q_k$, which is the permutation corresponding to the first $(2n - 2)$ j $+ k$ flips of the initial stack of coins, will be one of the "unlucky solutions." For example, if q_k = (I $\bar{2}$)(3 4 5 6)($\bar{1}$ 2)($\bar{3}$ $\bar{4}$ $\bar{5}$ 6) and the bottom of some column is 561234, then we will have all heads after k operations in the next column. Because the number of columns may be extremely large, it is better to look at the cycles of p than to generate the configuration at the bottom of each column.

AIS interpreted the desired goal as finding a transformation that moves from column to column in a fixed row, i.e., generating the permutations

$$q_k, pq_k, p^2 q_k, p^3 q_k, \ldots,$$

and detecting which ones are all heads. DEK noted that last time WC solved this for the special case of the bottom row (the solution was to find the minimum cyclic shift for each cycle that would make it all heads), and now we must generalize this. If p contains a cycle (x_1 52 53 54 x_5 x_6), then p^j will contain the cycle $(x_j\ x_{j+1}\ \ldots\ x_{j+5})$.

TPM pointed out that the permutation $q_k^{-1} p q_k$ represents the transformation from one column to the next along a fixed row. If we first use q_k to get to this row in the first column, and then make j applications of $q_k^{-1} p q_k$, the resulting permutation is

$$q_k (q_k^{-1} p q_k)^j = p^j q_k,$$

which is exactly what we determined it should be. By computing the cycle decomposition of $q_k^{-1} p q_k$, we should be able to find the all-heads solutions.

At this point, several different notations had been introduced, and the class spent some time deciding what to adopt as a common notation. The final choice was to use a form in which pq means "first apply p, then q," and in which cycles read from left to right.

DEK mentioned that he always gets mixed up when deciding on notation for permutations, since there are two posibilities depending on whether you consider the permutations of the positions or the contents of the positions; you have a fifty-fifty chance of getting it wrong, and in this case we seemed to have gotten it wrong on Tuesday. To get the correct interpretation we have to go more slowly. The question is whether the operation p takes the coin pattern $x_1 \ldots x_n$ into $x_{p(1)} \ldots x_{p(n)}$ or if it takes $x_{p(1)} \ldots x_{p(n)}$ into $x_1 \ldots$ x,, where p takes j into p(j). Under the first interpretation the operation p followed by q would take $x_1 \ldots$ x, into $x_{p(q(1))} \ldots x_{p(q(n))}$; under the second interpretation it would take $x_{q(p(1))} \ldots x_{q(p(n))}$ into $x_1 \ldots x_n$. So we should use the *second* interpretation to get our desired conventions! If the bottom of the first row is 561234, meaning that $x_1 x_2 x_3 x_4 x_5 x_6$ has been converted to $\bar{x}_5 \bar{x}_6 x_1 x_2 x_3 x_4$, the permutation p should be represented in cycle form as (1 3 5 $\bar{1}$ $\bar{3}$ $\bar{5}$)(2 4 6 $\bar{2}$ $\bar{4}$ $\bar{6}$). On Tuesday we had used (1 $\bar{5}$ $\bar{3}$ $\bar{1}$ 5 3)(2 $\bar{6}$ $\bar{4}$ $\bar{2}$ 6 4), since position 1 was filled with the complement of position 5, etc., but this convention doesn't multiply correctly; the proper form "(1 3 . . .)" says that the element in position 1 *moves* to position 3, etc.

DEK decided to try a test case in which $p =$ (1 $\bar{2}$)(6 5 4 3)($\bar{1}$ 2)($\bar{6}$ $\bar{5}$ $\bar{4}$ 3) and q_k = (1 $\bar{5}$ $\bar{3}$ $\bar{1}$ 5 3)(2 $\bar{6}$ $\bar{4}$ $\bar{2}$ 6 4) (although these might not actually occur in the G-coin problem).

Then

$q_k^{-1}pq_k$

$$= (1\ 3\ 5\ \text{i}\ \bar{3}\ \bar{5})(2\ 4\ 6\ \bar{2}\ \bar{4}\ \bar{6})\cdot(1\ \bar{2})(6\ 5\ 4\ 3)(\bar{1}\ 2)(\bar{6}\ \bar{5}\ \bar{4}\ \bar{3})\ .\ (1\ \bar{5}\ \bar{3}\ \text{i}\ 5\ 3)(2\ \bar{6}\ \bar{4}\ \bar{2}\ 6\ 4)$$
$$= (1\ 4\ 3\ 2)(5\ \bar{6})(\bar{1}\ \bar{4}\ \bar{3}\ \bar{2})(6\ \bar{5}).$$

DEK asked whether anyone knew a quick way to compute the cycles of $q^{-1}pq$, given p and q, and JF answered that the cycles of $q^{-1}pq$ are obtained by applying q to each element in the cycles of p. If we rewrite the result of the above computation as $(\bar{5}\ 6)\ (4\ 3\ 2\ 1)\ (5\ 6)\ (\bar{4}\ \bar{3}\ \bar{2}\ \text{i})$, we see that this is the case. DEK pointed out that we can apply this fact to our problem by not actually computing $q_k^{-1}pq_k$ explicitly; for most values of k, it will be possible to conclude that row k has no solutions without examining more than a few elements of p.

Continuing his presentation, TPM said that once we decompose $q_k^{-1}pq_k$ into cycles, we get a separate linear congruence for each cycle, and if these have a simultaneous solution, we have an all-heads configuration in row k. JF asked why there couldn't be more than one congruence per cycle, and DEK explained that this was impossible; he encouraged the class to prove this for themselves, over the weekend, pointing out that it is fortunate that at most one congruence per cycle is possible, otherwise the problem would be much harder.

Suppose that $q_k^{-1}pq_k$ has a 7-cycle that, when applied to the configuration in the kth row of column 1, produces all heads after 3 applications, so $q_k(q_k^{-1}pq_k)^3$ is an all-heads permutation; and there is also a 9-cycle that produces all heads after 2, 5, or 8 applications. The congruences to solve in this case would be

$$j \equiv 3 \pmod{7}$$
$$j \equiv 2 \pmod{3},$$

and it turns out that these are equivalent to the single congruence $j \equiv 17 \pmod{21}$.

Some systems of congruences may have no solution in integers, which means that there is no "unlucky solution" in that row. For example,

$$j \equiv 2 \pmod{6}$$
$$j \equiv 3 \pmod{9},$$

has no solution, since both $j - 2$ and $j - 3$ have to be multiples of 3 if $j - 2$ is a multiple of 6 and if $j - 3$ is a multiple of 9.

In general, if we want to solve the system of congruences

$$j \equiv a_1 \pmod{n_1}$$

$$j \equiv \text{a,} \pmod{n_r},$$

we can use the fact that a pair of congruences $j \equiv a \pmod{m}$ and $j \equiv b \pmod{n}$ is either unsolvable or is equivalent to a single congruence $j \equiv c \pmod{\operatorname{lcm}(m, n)}$. Applying this to the system of congruences one pair at a time, we can eventually get a contradiction or just one linear congruence. DEK encouraged the class to find a constructive proof of this fact, leading to an efficient algorithm. The special case $a_1 = \cdots =$ a, $= 0$ is what we solved previously, by computing the lctn of the n_i.

Class notes for Tuesday, October 13

DEK asked what results, if any, people had gotten so far. A few groups were able to present some of the numbers they had obtained, and some interesting patterns were discovered.

First of all, JF conjectured that "unlucky" or "sporadic" solutions exist for n = $3, 6, 10, 15, 21, \ldots,$ namely for all n that are triangular numbers. A triangular number is a number of the form

$$1+2+3+\cdots+(k-1) = \binom{k}{2},$$

which is so named because it is the number of points that can be placed in a triangular configuration, with one in the first row, two in the next, and so on. However, some values of n that are not triangular numbers were also discovered to have sporadic solutions. In fact, the largest number of sporadic solutions in the range tested is at n = 7, where five were found.

YOM noticed that sporadic solutions for triangular numbers were found near the end of the period, and when we examine exactly how far from the end they are,

n	sporadic solution	period	difference
3	3	8	5
6	53	60	7
10	531	540	9
15	437	448	11
21	1267	1280	13
28	1443	1458	15

we see an obvious pattern: the distance seems to be an arithmetic progression. KPB suggested that the case n = 1 be undefined, since $2n - 2$ flips is not meaningful. DEK agreed, but said that in developing a mathematical theory it is often useful to make statements that work for trivial cases such as $n = 0$ or n = 1.

YOM went on to say that when n is triangular, the configuration of the coins at the end of the entire period is 123456 . . . , i.e., the same as in the original state, and at the sporadic solution it is . . . 456231, which shows an interesting pattern. DEK suggested looking at the last seven flips in the case n = 6 to see why this is so.

	$\mathbf{4\underline{56231}}$
Flip 6 coins	$\overline{1}32\underline{654}$
Flip 3 coins	$231\underline{654}$
Flip 6 coins	$4\underline{56132}$
Flip 4 coins	165432
Flip 6 coins	$\underline{23456\overline{1}}$
Flip 5 coins	654321
Flip 6 coins	123456

If we go from bottom to top, and look at two steps at a time, we see that first the leftmost coin is flipped and put at the right end, then the left two coins are flipped and put at the right end, and so on. If n is a triangular number, then, we will get to an all-tails state this way, and the state before it will be all heads.

Last week, we saw that a sequence of three flips, namely (flip n, flip ***k,*** flip n), is equivalent to "flip the bottom k." This enables us to understand the first ***n/2*** configurations easily. Now we see that the flips near the end of the sequence also have such an interpretation, and these permute the coins in an easily-defined way for the last $\sqrt{n}$ steps. For the stages between $n/2$ and n-- $\sqrt{n}$, however, there seems to be no easy way to describe the configurations of the stack of coins. JLM showed the class a diagram of heads and tails configurations, which indicated that these stayed fairly regular up to about $\frac{2}{3}n$.

DEK mentioned that the triangular numbers provided an example of an infinite class of values of n for which there exist sporadic solutions to the coin-flipping problem. AVG described another class of solutions-those that occur when a power of the permutation p gives an all-tails configuration, since this must be immediately preceded by an all-heads state. No one could come up with a way of determining which values of n had this property. JF said that n = 7 has sporadic solutions at 75 and 80, which also remain unexplained.

The discussion now turned to the running time of the algorithm; it was agreed that as implemented by everyone so far, the time would be $O(n^2)$. This is because in computing p, O(n) time per flip is taken for 2n — 2 flips, requiring $O(n^2)$ time; the operations on the cycles of p are easily seen to be bounded by $O(n^2)$; and the search for unlucky solutions takes O(n) time in each of the 2n — 2 rows. DEK asked if anyone had thought of algorithms which would run in better time that $O(n^2)$, but no one had. He suggested that a "divide and conquer" method might be appropriate, and in answer to a question, explained that this means reducing a problem of size n into two problems of size n/2 and combining the solutions in time O(n). If this can be done recursively to the subproblems, then the total time will be O(n log n). (For further explanations of this technique, see Aho, Hopcroft, and Ullman, *The* Design and Analysis of *Computer* Algorithms, pp. 60-67, and J. L. Bentley, "Multidimensional Divide-and-Conquer," *CACM*, April 1980.)

It is not clear how to apply divide-and-conquer to the penny-flip problem, but DEK mentioned that Jim Boyce (an alumnus of CS 204) came up with an efficient strategy. We can use a balanced tree to represent a configuration of coins, and at each interior vertex have a bit that tells whether the subtree rooted at that vertex is in correct order or reversed. Then each of the "flip ***k*** coins" operations can be done by an appropriate combination of tree-splitting, reversing, and tree-merging operations. The reversing is done just by changing the proper bit (i.e., not actually moving any objects), and algorithms for splitting and merging in O(log n) time are known. Therefore, the total time to do the $2n - 2$ flips is 0 (n log n). Implementing this, though, would be much harder that the more direct algorithm developed so far.

However, this does not help in Ending the "unlucky" solutions, and so far no one has come up with an algorithm for that which is better than $O(n^2)$ in the worst case. DEK mentioned that what could be done is to take advantage of the fact that sporadic solutions are relatively rare, and instead of completely decomposing $q_k^{-1}pq_k$ into cycles, we could do this "virtually" and generate only the values as they are needed. When we are working in the kth row, the probability that we can continue searching for a solution is $\frac{1}{2}$ after one step, $\frac{1}{4}$ after two steps, etc., and most of the time we conclude that there are no unlucky solutions in the row after only a few steps.

After class, YOM and LAH described a method for solving simultaneous congruences that they were using in their program. Suppose we are given the two congruences

$$j \equiv a_1 \text{ (modulo } n_1),$$
$$j \equiv a_2 \text{ (modulo } n_2).$$

Let $g = \gcd(n_1, n_2)$. The extended gcd algorithm computes integers x_1 and x_2 such that $g = x_1 n_1 + x_2 n_2$.

A necessary and sufficient condition for there to be a solution is that g divide $a_2 - a_1$, i.e., $a_2 - a_1 = sg$ for some integer s. Now, modulo $\operatorname{lcm}(n_1, n_2)$, we have

$$j \equiv a_1 + b_1 n_1,$$
$$j \equiv a_2 + b_2 n_2$$

for appropriate b_1 and b_2. Subtracting, we get $a_2 - a_1 \equiv b_1 n_1 - b_2 n_2$, and hence $sg \equiv b_1 n_1 - b_2 n_2$. Since $sg = s x_1 n_1 + s x_2 n_2$, we have $b_1 \equiv s x_1$ (and $b_2 \equiv -s x_2$), so the desired congruence is

$$j \equiv a_1 + s x_1 n_1.$$

After computing the gcd, we can find s in one step, and readily have the solution for the combined congruence.

Solutions for problem 1

All of the groups wrote programs based on the method discussed in class. The results for $1 \le n \le 120$ were the following:

n	t_k	Sporadic Solutions	n	t_k	Sporadic Solutions	n	t_k	Sporadic Solutions
1	2		41	51040		81	69120	
2	8	3	42	11480		82	3720816	
3	8	3	43	16128		83	66912	
4	36	17	44	450296		84	3133914	
5	40		45	88704	88685	85	11684736	
6	60	32, 53, 27	46	4860		86	2917200	
7	144	122, 75, 18, 80, 117	47	15088		87	4608912	
8	196	97	48	518880		88	4478760	
9	128	43	49	43776		89	164736	
10	540	531	50	25284		90	149520	
11	200		51	82000		91	190080	190053
12	440	219	52	5100		92	753480	
13	312		53	73008		93	287776	
14	2184		54	2270520		94	156240	
15	448	437	55	517104	517083	95	135360	
16	2700		56	23100		96	4806240	
17	2560	501	57	239904		97	318720	
18	1088		58	970368		98	38024	19011
19	15120		59	1913072		99	38024	
20	608		60	276120		100	540540	
21	1280	1267	61	12960		101	11414400	
22	1848	923	62	266448		102	2188872	
23	1848		63	461280		103	12631680	
24	828		64	100548	50273	104	36165360	
25	2112		65	6912		105	557232	557192, 557203
26	1500		66	107640	107617	106	4520880	
27	1144		67	137808		107	549504	
28	1458	1443	68	273360		108	41944	
29	2688		69	404736		109	64800	
30	51040		70	513912		110	47088	
31	12000		71	856800		111	13877600	
32	5208		72	93720		112	4307688	
33	14336		73	12096		113	5720064	
34	55440		74	367920		114	208824	
35	1904		75	37296		115	563616	
36	32760	32743	76	40500		116	676200	
37	8640		77	178296		117	551232	
38	163540		78	55440	55415	118	24189984	
39	20064		79	773136		119	4125423040	
40	31122		80	663600		120	26860680	26860649

Several groups discovered improvements to the algorithm, or studied some of the details of the problem in greater depth. These results will be described here.

Using the fact that (flip *n*, flip *k*, flip n) is equivalent to "flip the bottom ***k*,"** *we* can simplify the computation of the permutation p. **RJA/AVG called** this transformation "rflip *k*," and noted that the sequence

flip 1
rflip 2
flip 3
rflip 4

would generate rows $1, 4, 5, 8, \ldots$ of the original configurations. However, since every even-numbered original operation is a "flip n," an all-heads state in one of the missing rows will correspond to an all-tails state in one of the rows that is generated. RJA/AVG decided that finding the all-tails states would complicate the program, and did not use this method. However, DMC/ JLM, making the same observation, decided to test for all-tails configurations and use only half of the rows.

LAB/YOM discovered a data representation that sped up the computation of p. This was done by performing each "flip n coins" operation "virtually." In their representation of a permutation, a flag tells whether the state is normal, or "upside-down," i.e. with left and right reversed, and heads and tails interchanged. Flipping the whole pile requires simply complementing this flag, without moving data in any arrays. If n is even, the flag is in the "upside-down" position after the final flip, so some extra work is needed to normalize the position. But the time for the whole operation, originally estimated as $\frac{3}{2}n^2$ basic operations, has been reduced to approximately $\frac{1}{2}n^2$. LAH/YOM also used a more compact notation to represent permutations in cycle form, but switched to the representation used in class for the shifting operations needed in finding the congruences.

Some groups, including DMC/JLM, RJA/AVG, ARS/JCM/JF, and DEF/DPH/TPM, noted that when given a pattern of heads and tails, the minimum shift (with wrap-around) that preserves the pattern must be a divisor of the length of the pattern. The name "pseudoperiod" was given to this number by some of these groups.

ARS/JCM/JF said that when combining congruences, it would make sense to first try those arising from shorter cycles, in the hope that a contradiction could be found with less work. This could be done by sorting the cycles in increasing order of their pseudoperiod. AH/AIS found that longer cycles were usually discovered first in their program, so they used the cycles in the opposite of the order in which they were generated to solve congruences. DEF/DPH/TPM said that they were unsure whether analyzing long cycles or short cycles first would be better.

KPB noticed in a debugging run that for most large n, the nonexistence of a solution in row k was discovered in the analysis of the first cycle of $q_k^{-1}pq_k$, and the solution of simultaneous congruences was then not needed.

LAH and YOM discovered a "parity argument" that reduced the total amount of their computation significantly. Let a coin that is "heads" have parity 0, and let "tails" be 1. The parity of a configuration of n coins is the sum of the coins' parities. We are only interested in whether this number is even or odd.

The first thing to notice is that flipping an even number of coins conserves parity, while nipping an odd number changes it. Now, consider the change in parity as a result of the permutation p (the first $2n - 2$ moves). If p is even, then $q_k^{-1}pq_k$ will be even for all k, which means that moving along row k will not change the parity of a configuration. So if we start with an odd state in the first column, all columns of row k will be odd, and we can never reach an all-heads configuration! Therefore, we need only test the rows that start with an even state.

On the other hand, if p is odd, then each $q_k^{-1}pq_k$ is odd, meaning that as we move along row k, the parity of the configuration alternates between odd and even. Thus we know that an all-heads position can only occur in an odd column or an even column, depending on the parity of the first column of row k. However, in computing with the cycles of $q_k^{-1} p q_k$, it is still possible for some cycles to reach an all-heads state in any column. Therefore, LAH/YOM used the parity information in their program only by introducing a congruence $j \equiv 0 \pmod 2$ or $j \equiv 1 \pmod 2$ into the set of simultaneous congruences, rather than ruling out the use of half of the columns entirely.

The parity of p is, moreover, easy to find. It is even when n is congruent to 0 or 1 modulo 4, and odd when n is congruent to 2 or 3.

The infinite class of sporadic solutions corresponding to triangular numbers was discussed in class. **ARK** noticed that an all-tails configuration at position $\frac{1}{2}t_k$, preceded by an all-heads configuration, occurred for $n = 2^1, 2^3, 2^6$, as well as some other values of n. She conjectured that this happens for $n = 2^k$, whenever k is a triangular number. (Nobody ran their program at $n = 1024$ to test the next case in this sequence.)

DMC/JLM ran their program for $2 \leq n \leq 400$, and for some larger values, but said that solutions for large n are unreliable since single-precision arithmetic was used. They also collected timing statistics, which agreed with the $O(n^2)$ running time derived from analysis of the algorithm.

Problem 2: Archival Files

Consider a file F_0 that has just been created and put onto a file server, where many people can read it. Later on the file is updated by a series of operations U_1, and we obtain a new file $F_1 = F_0U_1$. Eventually F_1 changes to $F_2 = F_1U_2$, and this process keeps going until we reach the nth version, $F_n = F_0U_1U_2 \ldots U_n$.

Sometimes it is important to be able to get at any of the versions F_j of a file, not just the most up-to-date one. For example, F might contain software that has been distributed to a large number of customers; most customers have not installed the latest version and you might need to recreate their environment when you receive an emergency phone call. Alternatively, some other file might refer to F, and the reference may be meaningless unless it is to the particular version of F that was current when the reference was first made.

On the other hand, it is usually too expensive to retain all versions of F, since this rapidly fills up one's disk allocation.

Therefore it would be nice if operating systems would provide a special means of handling such files. Let's suppose that when a file F has been declared "archival" the system will treat it in a special way that makes all versions accessible without excessive running time, yet the total storage space should be reasonable.

Your problem is to design a good way to represent and manipulate such archival files. This is an open-ended problem, part of which consists of deciding what it means for such a system to be "good." You might want to consider, for example, whether it is advisable to store a few versions from which others can be reconstructed, noting that $F_{j-1} = F_jU_j^{-1}$ so that updates can be undone. Does a "divide and conquer" approach lead to a method that provides logarithmic overhead? Try to find a scheme that has a good tradeoff between space and time.

We shall assume that the files we deal with are text files, i.e., long sequences of characters, and that "update" operations consist of one or more transformations that replace one contiguous substring of a file by another string. You may wish to make further assumptions, e.g., that each update changes only a small percentage of the file, or that F_j will probably be "closer" to F_{j-1} than F_{j+1} is; however, you might also be able to think of a scheme that works well without these assumptions. One way to test your methods might be to apply them to the transcript files written by the BRAVO editor.

Class notes for Thursday, October 15

DEK started the discussion of the file archival problem by mentioning that it has recently become one of the more important problems in computer science. As computers are used to hold large databases whose contents affect millions of people, efficient solutions to problems such as the one we are considering are needed. Another application, and the one from which this problem originated, is the on-line storage of large text files, such as the text of DEK's book *The Art of Computer Programming*. The general problem that is being dealt with here involves a tradeoff between time and space.

KPB asked whether we could limit the problem to handling text files. DEK pointed out that it does not really matter, since any file is just a collection of bits, and one that is organized in words could be considered a stream of characters over a very large alphabet. However, such a generalization does not necessarily mean that algorithms that are suitable for small alphabets (such as a case statement with one case for each character) will also be useful for all files. AVG suggested that dealing with character files is harder than working with files organized as sequences of records, and that when that sort of format is used, a program can keep a log listing new records inserted and changes or deletions of records. DMC added that a variable-length record format would make the problem harder than a fixed record structure. DEK said that the archival problem in these cases is indeed simpler, and file storage did not become an area of increased research interest until there was a greater demand for archival of text files that are not organized with a record structure. In a text file, the ordered sequence of characters is significant, while in the other case it is the unordered set of records that is relevant.

AVG continued by saying that in a record-based system, there is often the concept of a "transaction ," a basic operation that affects one record, and that a log of these transactions is easily kept by programs that modify the file. AIS noted that an entire editing session with a text file can be considered a "transaction." JLM pointed out, however, that when a program makes changes to an entire file, there is no simple way of recording what has been done to the file. We cannot even say "run the program on the original file" to generate the updated file at some later time, since the program itself may be modified and may not then produce the same results. Therefore, we must find a method to describe changes to a file that is general enough to handle all kinds of changes.

Considering the case of modification by an editor, DPH suggested that saving the editor commands in sequence would provide the necessary information. The BRAVO text editor used on the ALTO computers keeps such a "transcript file." JCM gave a brief description of these files, and noted that they can only be used in the "forward" direction—to construct a new file version from an older one-since the text of deleted strings is not saved. Most of the class had not seen the BRAVO editor, so DEK described it briefly. AH said that saving the keystrokes of an edit session would preserve as much information as possible, but JLM noted that this was valid only as long as the editor itself didn't change. DMC added that the edit commands could include user-defined macros in many editors, and the macro definitions would have an even larger possibility of change. This problem could be avoided, though, by saving the editor commands into which the macro was expanded.

Another approach, suggested by GSK, is to use a file-comparison program to find the differences between two versions of a file. He said that in his experience, such a program generally worked quite well on most files. DEK said that file-comparison programs are often vital components of a computer system, and described how the program **SRCCOM** on the SAIL system is run. KPB asked whether there was any algorithm that could do this type of comparison in better than $O(n^2)$ time. DEK said that the best known algorithms are $O(n^2/\log n)$, and that work by A. Yao suggests that this is actually a lower bound on the amount of work needed. This problem also has applications in areas other than computer science; for example, biologists need to compare long strands of DNA efficiently.

Next DEK described a simple comparison algorithm that is $O(n^2)$ with an example. Suppose the first text string is the word mime and the second is smiles. We create an array with four rows and six columns, labeled by the letters in the two strings, and place a 1 in the array wherever two letters agree.

$$\begin{array}{c|cccccc|}
 & \mathtt{s} & \mathtt{m} & \mathtt{i} & \mathtt{l} & \mathtt{e} & \mathtt{s} \\ \hline
\mathtt{m} & & 1 & & & & \\
\mathtt{i} & & & 1 & & & \\
\mathtt{m} & & 1 & & & & \\
\mathtt{e} & & & & & 1 & \\ \hline
\end{array}$$

The object is to find a path from the upper-left corner to the lower-right corner that moves at each step either down, to the right, or crosses a 1 moving both down and to the right, and that crosses as many 1's as possible. (The points on this path are between the array elements; the upper-left corner will be called (O,O), and in the example, $(4,6)$ would be the lower-right corner.) If a_{ij} is the value (0 or 1) in the (i,j)th array element, and if $\ell(i,j)$ is the number of 1's on the maximum path from point $(0,0)$ to point (i,j), then $\ell(0,0) = 0$ and we can compute the other values of $\ell(i,j)$ from the equations

$$\ell(i,j) = \begin{cases} \max\bigl(\ell(i-1,j),\ell(i,j-1)\bigr) & \text{if } a_{ij}=0 \\ \max\bigl(\ell(i-1,j),\ell(i,j-1),1+\ell(i-1,j-1)\bigr) & \text{if } a_{ij}=1 \end{cases}$$

This method is, however, too slow to be used on files of any considerable length. In a text file, we might treat each line as a unit, and apply a hash function to the lines of the file. Then we could use the method just described to compare the hash values.

YM pointed out that a common operation in a text file is to move large blocks of lines (such as a paragraph), and that by recognizing diagonal blocks of 1's in an array such as the one above, we could make note of this fact rather than recording the change as a deletion of text together with an insertion of the same text somewhere else.

The discussion now turned to the question of what level of change to a file to consider. YM remarked that a multiple search-and-replacement command in an editor might change a file in many places, though it is only a single command, and DEK suggested that we restrict the set of transformations to be considered; hopefully those that we choose will be able to easily express the editing operations that people do most often. DPII asked at what level we should consider a new "version" of a file to be-in the extreme cases we could let each editor command create a new version, or we could let an entire editing session define a single new version of the file. DEK suggested that we incorporate something in between, so that the user of our system will be able to note when version n becomes version $n+1$. If we store n versions, our retrieval algorithm should hopefully be able to get to any of these versions in less than $O(n)$ time and/or space.

AVG asked if we had decided whether creating a log file of changes, or using a file comparison program to determine changes, was a better method. DEK pointed out that a transaction log is potentially less efficient, since multiple changes to the same part of a file might occur in going from one version to the next. JLM suggested that we try both methods, and see experimentally which one is better. DMC noted that the output from a file-comparison method is a set of "transactions" where there are only a few simple transactions possible, and suggested that we could output more complex operations that resemble editing operations more closely. JF pointed out that any editing operation can be reduced to insertions and deletions, so this is not really necessary.

Another question is whether all versions of the file are equally likely to be accessed. AIS suggested that if the most recent version is more likely to be needed, then we should take advantage of that fact and make that case more efficient. DMC said that a system could make a history of its requests, and possibly adjust the information stored as the requests change. JF added that we may want to record that some versions of the file correspond to major revisions, since these are more likely to be referenced.

RJM outlined a list of unanswered questions that pertain to this problem. Is the minimum object that can be changed a character, a word, a bit, etc.? Is there any relation between the characters in the file? If so, a change such as correcting the spelling of a word is easier to record. Is the file length fixed or variable? What statistical characteristics about changes to the file can be known in advance? DEK said that these are all important questions to ask when beginning research on a problem, such as this one, which is not completely defined. Then we have to fix some of the choices and stick with them, keeping the other options buried in the back of our minds.

Two extremes of the possible files that will be archived are a large file that has updates to only a small section at a time, and a small file that is subject to frequent major changes. AH pointed out that in the first case, it is more efficient to store the original version and a record of the changes that were made, while in the second case, it is easier to store copies of the entire file. TPM noted that saving only the changes uses the least amount of storage, while saving all versions gives the fastest access to any one of them. Some compromise between the two methods must be found.

Using the notation of the problem description, DEK noted that

$$F_k = F_j(U_{j+1} \ldots U_k) = F_j U_{jk},$$

where U_{jk} is defined to be $U_{j+1} \ldots U_k$, i.e., the sequence of changes that get us from version j to version k. We might want to store U_{jk} for some, but not all, values of j and k, and thereby be able to retrieve particular versions of the file more efficiently. We also have the relation $F_j = F_k U_{jk}^{-1}$, meaning that we can work backwards provided that the update information is stored in a form that is reversible. JCM said that if we keep several versions of the file, then when a new one is requested we can discard the least recently used one, and noted the analogy between this and LRU paging schemes in operating systems. There arise potential problems if someone updates a version of the file other than the most recent one, since then we will have versions of the file related to each other in a tree structure rather than a simple linear sequence.

JPM noted that in the two extreme versions of files mentioned above, the tradeoffs were not completely clear, because we can construct pathological cases. For example, the operations that produce one version of a file from the previous one may take arbitrarily much space (by repeatedly changing the information in some part of the file), and a comparison of the old and new versions would then be more efficient even though they are almost identical. He also suggested an extension of the problem to a situation in which the history of a file may include its being formed by merging together other files; we would want to be able to reconstruct versions of files that no longer exist from the information that is kept about this sort of updating operation. DEK said that this was an interesting idea, but suggested that the class stick to the original problem to keep the work manageable.

Class notes for Tuesday, October 20

DEK suggested that the class spend today discussing only one aspect of the problem, even though people might have different opinions about what the best way is to proceed. It pays to first solve a simple version of any problem; otherwise we surely won't be able to attack the most general version. As the basic model, DEK suggested a fairly large file that is subject to a sequence of relatively small updates. An example would be a dictionary.

In this case, JCM noted, we would be most likely to make additions to the file, and much less likely to delete anything or change the order of text. It is probably true of most text editing that addition is the most common operation. JPM pointed out that a dictionary could be organized as a sequence of records, and that each primitive operation would add, delete, or modify exactly one record. This is essentially what is done in manipulating a database. KC added that the entries in a dictionary contain information that describes their positions relative to other entries; this is not true of text files in general.

KPB asked whether we would be working with variable-size data items, or with a fixed-size record format, which would be easier. DEK said that it did not matter at this stage of the problem; we should deal with the files as abstractly as possible and make such decisions only when necessary.

DMC asked whether a division into records was necessary, since we can identify any character in a text file by just giving its position. DEK pointed out that the difference occurs when a record can be accessed using a key. Also, a database may be organized either as a set of records or as a *sequence* of records. In the first case, retrieval by key is necessary, while in the second case, we have to keep track of the order of records in the file, and can access records sequentially. These are two very different problems.

DPH had worked on the problem of determining the changes from one file version to the next, given the two versions for comparison, but noted that we now seemed to be assuming that the record of changes is already given. DEK said that this was really a separate problem from what we are now considering, and that we should concentrate on just the one problem. JF said that she didn't see why the two versions (file of records and file of characters) are separate, since a text file can also be viewed as a sequence of records; thus one view may be more powerful than the other. DEK said that if this were so, there might still be a way to solve the less general problem that is more efficient than a general solution. JPM pointed out why the one method is more general: in a text file, the only key is the implied key given by the position of each character, while with explicit records, we can make many choices for keys and for the sequence of records.

Regarding the difference between sequences and sets of records, DEK noted that it might be extremely hard to compare two files that were organized only as sets. In fact, the best method is probably first to sort the records, and then compare, which takes $O(m \log m + n \log n)$ time if the files have m and n records, respectively. If there is no ordering relation on the keys, even this method cannot work, and we may have to use order mn time. He asked the class to vote on whether we would look at sets or sequences of records; the majority of the class preferred sequences.

The discussion now turned to examining how to represent the information in the archival files. JLM presented a scheme, which the class decided to call an "amalgamated file," that contains the text of each record and a table that will enable us to reconstruct any of the versions of the file. Suppose we have four versions, containing the following records:

$$\begin{array}{lllllll} F_0 & = & a & b & c & d & e \\ F_1 & = & a & f & c & d & e \\ F_2 & = & a & f & C & g & d & e \\ F_3 & = & h & f & g & d & e \end{array}$$

We store the text of all records that were ever in the file, and assign each one a number. For example,

1	2	3	4	5	6	7	8
h	*f*	*g*	*d*	*e*	*a*	*b*	*c*

The lines have been arranged so that the current version of the file consists of the lines at the beginning. In the table, we have a set of lists that tell us the history of each line. For the set of files shown above, the table would be

(1)
(2 2 2)
(3 4)
(4 5 4 4)
(5 6 5 5)
(6 1 1 1)
($\bar{7}$ 0 0 2)
($\bar{8}$ 3 3 3)

The interpretation of these lists is that the first element names the current position of the corresponding record (with a special notation, such as $\bar{6}$, for records that have been deleted), the next is the position in the previous version, and so on. The list ends at the point where the record was first inserted into the file.

Several ways were suggested to abbreviate this notation. If some text had never moved at all, it does not have to be mentioned in the table. Also, long sequences of the same record number can be compressed. DEK commented that for each proposed abbreviation, we must be sure to check that it keeps the table compact, makes the table easy to update, and does not complicate retrieval of any version of the file. Since insertion of a record is a common operation that alters a lot of the numbers in this notation, it will be important to find an abbreviation for that. RJM suggested that we store the relative position of each record instead of the absolute numbers shown above. JCM noted, however, that this would make it more difficult to represent the swithing of groups of records. KPB pointed out that this is never a problem in the example of a dictionary, since once records are in order, they will never be rearranged.

RJM continued to say that storing the relative position changes the distribution of numbers stored in the table-the number ***0*** would probably be most common--and we could use a Hamming code to compress the information. This approaches the minimum theoretical limit for the amount of storage needed. DEK pointed out that encoding the table would complicate the retrieval and update processes.

JLM suggested combining groups of lines that move together; thus in the above example, **(3. .5 4.** .6) could represent moving three lines in going from F_3 back to F_2. He foresaw potential problems with this method, though. AIS added that some compression schemes such as this would not enable us to get the minimum access time.

DPH presented a method that would have access time that is linear in the length of the amalgamated file. This is done by including the version numbers of the first and last appearance of each record, and ordering the records so that each version can be scanned in proper sequence. In the example, we would store

$$\begin{array}{cccccccc} 0 & 3 & 0 & 1 & 0 & 2 & 0 & 0 \\ a & h & b & f & c & g & d & e \\ 2 & \infty & 0 & \infty & 2 & \infty & \infty & \infty \end{array}$$

DPH noted that the size of the table is also guaranteed to be bounded by a constant times the number of records. AI-I said that if we use a structure with pointers, we can also change the order of blocks of records. This requires, though, that the pointers themselves have an associated history. YOM observed that pointers would be necessary to handle insertions in this type of data structure; otherwise we would have to move most of the records themselves.

A combination of this with the previous method was proposed by RJM. We can use a table of lists as before, but with relative position information, so that in the example above, we would have

(1)
(2 0 0)
(3 1)
(4 0 0 0)
(5 0 0 0)
.

The offset of any record is gotten by keeping track of what has been done before, so that the "1" in the list for record 3 means all records from 3 on move by **1. Now, we compress** the multiple ***0's,*** so, for example, record 4's list will read **(4** (0 3)). Since each record has a simple history (it is born at some time, lives for a while, and may eventually die), the size of this list is always quite small. (However, this is not true if we consider swapping records as a primitive opration.) JPM pointed out that this scheme loses some generality, because the amalgamated file must be scanned from its beginning in order to find the position of any record in any version.

DEK summarized what had been presented in today's class. We have two methods that are possibly equivalent. It is generally accepted that a good running time is one that is linear in the size of the amalgamated file. If we are willing to use a line of text instead of a single character as our basic record, then the extra storage overhead of pointers and of numbers giving the histories of lines is reasonable. He also suggested that the class think about what sort of data structure best represented the operation "if the version number is greater than a and less than b then . . . else . . . ," since this seems to be a crucial operation in the retrieval process.

Class notes for Thursday, October 22

Today the class considered some of the more concrete aspects of solving the problem. KPB began by asking what kind of input people thought should be used. AH said that for testing the program and illustrating the algorithms, it would be sufficient just to use single characters. Later one could worry about the problems associated with operating on actual files.

DEF suggested creating a simple editor that has commands to add, replace, and delete text from a file. RJM pointed out that this approach would allow access to a large amount of test data, since any text file could be used as input. The class agreed that this was a more feasible idea than using the BRAVO transcript files, since most were unfamiliar with BRAVO. AVG noted also that with a mini-editor, it would be easy to check the results of the program. DEK pointed out that it would be neat to use such a system on the writeup for a group's solution to problem two, so that the grader could look at all the versions of the writeup.

DEK asked whether there was a way to produce test data randomly. For the method of comparing old and new versions to compute the changes, TPM said this would be hard; DMC suggested that you could produce a random new version by generating a sequence of commands and applying them to the existing version.

JF thought that it was unnecessary to create an editor to solve this problem, since it is possible to create both the files and the update information by other means. DEK agreed with this, but did not discourage anyone who wanted to use the editor approach from doing so, if this was how they best viewed the problem. He mentioned that approaching a problem theoretically and writing actual programs are both useful approaches, and that he tries to do both whenever possible. Someone who works purely on the theory might find a theoretical solution that is linear in file size and logarithmic in the size of the updates, by storing the updates in a tree structure. This would not be considered useful by a programmer, since in most applications the size of the updates is small and doesn't justify the extra effort. When the theory in an area of research gets too far away from the applications, it often starts to become uninteresting.

KPB now outlined a division of the solution into three programs: `ARCHIVE`, which would convert a normal text file into an amalgamated file with only one version; `EDIT`, which would add version F_{n+1} to an amalgamated file having versions $F_0, \ldots, F_n$; and `RETRIEVE`, which would extract any version F_k and present it as an ordinary text file. JCM pointed out that if one also wants to be able to update a version other than the most recent, then the last step is best split into two: one that produces an amalgamated file $F_0, \ldots, F_k$ from $F_0, \ldots, F_n$, and another that extracts F_k from $F_0, \ldots, F_k$. If we use the editor approach, JEH noted, we would have to extract version F_{n+1} as it was being produced, to be able to see it.

A question about the format of files organized into lines was brought up by AVG, and DEK described the standard format that is used on the SCORE and SAIL systems. Each 36-bit word contains five 7-bit bytes holding ASCII-coded characters, with the rightmost bit unused. A line ends with a carriage-return character (octal 15) followed by a line feed (octal 12). A page in a file is marked by a form-feed (octal 14). Also, any bytes that are all 0 are ignored by most programs. He explained that more modern systems do not require an end-of-line character or characters to separate text into multiple lines on a screen; for example, BRAVO does its own line-breaking and its files only mark the end of each paragraph. This is because text-processing systems are document-oriented, while general computing systems are more often line-oriented.

AIS asked whether we should assume that archival files may be too large to completely be read into main memory. DEK said this might be a good assumption, if you could do everything by sequential access to files. This caused some disagreement-several people felt that with an operating systems that provides paged memory, it might be best to let the system handle input/output in whatever way it chooses. DEK argued that the choice of who should do the I/O depends on the nature of the problem. If access is sequential, using system resources such as UNIX's pipes is called for, while if random access is needed, one shouldn't hesitate to read in the whole file. He added that there are certain applications, such as sorting, where the best algorithm is very dependent on the operating system's paging strategy. At any rate, it is a good idea to stick to sequential access if you can do so without serious loss in the generality of your algorithm.

AVG noted another problem with file size-there might be a case where the archival file is so large that we cannot create a second copy, even temporarily. This is probably beyond **what needs** to be considered in this problem, though JLM added that often a simple program becomes a part of a much larger environment, and then some limitations are carried on and make it awkward to use.

DEK presented an idea that had not yet been discussed much: interpreting the data in the amalgamated file as a "program" in a specialized language. This causes the archival process to become more decentralized, since some of the control information is contained in the files rather than in the programs that use them. Last time, we associated two version numbers a and b with each record, and had this represent "if the version number is between a and b, then include this text, else don't." If we think of this as an "opcode" with a and b

as the operands, then we can generalize to other opcodes. For example, a "jump" operation would allow us to skip over large blocks of records without scanning them individually. This would be useful for a small file with many updates. DMC suggested that the text in a line could be thought of as one of the operands to a command.

The class ended with some discussion of record-oriented files. KPB was interested in the dictionary concept introduced last time, and said that such a file could have a two-part structure. Each record would have a name and its contents, and there would be information about when it was inserted and deleted. AIS said that some IBM database systems have such an organization-records never are actually removed from the database. DMC asked whether this would obviate the need for using pointers. DEK noted that there are a lot of questions of efficiency in accessing such a database. JLM presented an interpretation of multiple versions as a generalization of fonts: just as a file might have some text in normal type for everyone to read and some in "fine print" for a special set of users, the different versions of an archival file could include text that pertained to some users of the file and not others. DEK commented that in such a scheme, a "version number" is likely to be more complex than a member of a simple linear sequence.

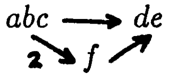
Class notes for Tuesday, October 27

Today the class finished discussion of the file archival problem, and described what they had done in their programs. RJA's group had used essentially the method described in class, keeping "birth" and "death" information for each record, and inserting records into the proper position so that only a single pass of the file is necessary to do retrieval. JCM had added to this scheme operations that could move large blocks of lines at a time. JSW asked whether a method that moved a lot of records for each insertion could be improved upon, but RJM thought this did not really matter, since an update involving an insertion of a single line would still require reading and writing the entire archival file to create the new version. DMC said that random access to a file is needed if you want to record only the changes and minimize the time.

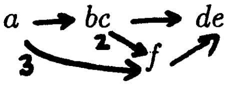
One way to cut down this overhead, which RJM used, is to maintain the text of the records separately from a graph containing the control information. DPH used a tree structure with blocks of unchanged text at each node, and YOM pointed out that this extends easily to a non-linear sequence of files, so that one can edit versions other than the most recent.

Before starting the main work on his solution, JLM said that he had used LISP to experiment with various ways to manipulate strings. KPB had also done some preliminary work, by looking at the "fonts" generalization that was mentioned last time in class.

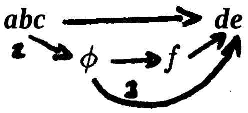
Returning to a discussion of the representation of archival files, AH pointed out that by keeping a separate "pointers" file, a simple scheme might cause the number of pointers to grow arbitrarily large; for example, when records are switched and then switched back without changing their text. DMC noted that by keeping only the pointers for actual changes in the file, this problem could be solved. Someone asked for an example of the use of pointers, and RJM showed how his group's representation worked. If the initial sequence of records is, say, *abcde*, and the first update changes this to ***nbcjde,*** then the information stored can be represented pictorially as



(The number 2 next to one of the pointers names the first version that uses that pointer.) If version 3 is ***ujde,*** then the pointer structure is changed to



If, however, version 3 is *abcde*, then the amalgamated file would be represented as



The record labeled ϕ is one that contains no text; it exists only to allow the proper placement of pointers.

JSW asked what other ways people had used to record changes to an archival file. DPH said that one method of optimization is to detect when a line has been both inserted and then deleted while producing a new version, and then not keeping the record in the amalgamated file. DEF said that since at the end of an editing session, all the changes can be found in one pass over the new version, it is reasonable to make also one sequential pass over the archival file to record the changes.

TPM had recorded the exact time of each change in his program, and **LAH pointed** out that this could be extended to a system that could allow one to "back up" in the middle of an editing session to any arbitrary time.

Solutions for problem 2

Most groups solved this problem with programs to convert an ordinary text file into an archival file, to update archival files, and to extract the text of a version into a text file. For the sake of consistency, these operations will be called CREATE, UPDATE, and RETRIEVE in this summary.

The UPDATE operation took the form of a sequence of editing commands. RJA/DEF/AIS read in these commands from a separate file, and said that they could be assumed to be the output from a file comparison program. Other groups wrote miniature editors, which took commands from a terminal and immediately changed the archival files (or kept the commands in memory, and produced a new version at the end of an editing session). KPB/TPM, ARK/JLM/JCM, and DPH/LAH/YOM wrote line-oriented editors.

DMC/JF/JEH wrote a character-oriented editor, whose interface to the user contained two pointers called the cursor and the *marker.* A mark command was used to place the marker at the current cursor position; then the cursor could be moved, and the delete command operated on text between the marker and the cursor.

In addition to insert, delete, and commands to read or write versions of the archival file, there were commands such as

- undo: reverse the effect of the most recent deletion (ARK/JLM/JCM).
- move: transfer text in the file without doing a deletion followed by an insertion (ARK/JLM/JCM, AH, DMC/JF/JEH). This was accomplished in the DMC/JF/JEH program by putting the location of deleted text into a "register" internal to the program, and then providing a command that inserted the text pointed to by the register at any desired location. A result of this was the ability to create multiple copies of text with only one copy of the characters actually in the archival file.
- replace: replace a line with new text (KPB/TPM).
- back: return to a previous version of the archival file (ARK/JLM/JCM). This was paired with a command to retrieve the most recent version of the file. By making this division, their approach was to concentrate on making the most recent file version the easiest to recover.
- simplify: Create a new archival file with only the active lines in the current version (ARK/JLM/JCM).

Some groups also had commands to move around in the file (rather than requiring the user to give line numbers with each command), and to type parts of the file on the terminal.

There was a variation in the methods used to store and access the information in the archival file. RJA/DEF/AIS kept a "birthday" and "deathday" version number with each record, and kept the records in order so that one pass through the archival file was needed for each update or retrieval. AH kept two files: a "text file" and a "pointer file." Lines were added to the text file only at the end, and the pointer file, which was small enough to read into main memory during execution of the program, contained a sequence of pointers in the proper order, each having a first and last version number. This scheme allowed transferring records as well as insertion and deletion. DMC/JF/JEH used a similar approach. When the pointers were read in, they were stored in a doubly-linked list to facilitate moving the cursor in both directions.

KPB/TPM stored everything in one file, including the pointers in a doubly-linked list, and used random file access. Random access was also used by DMC/JF/JEH. This allowed their retrieval time to be linear in the length of the version retrieved plus the length of the control file. KPB/TPM noted that random access could create some inefficiency, depending on the access method provided by an operating system, if groups of logically contiguous lines became widely separated in the archival file. Their proposed solution was to reorganize the file during periods of light load on the computer, to something logically equivalent but with records in a better order.

RJM wrote a program that maintained a text file (of single characters) and a file of pointers using the graph structure that he presented in class. His original decision was to allow any node to point to any number of other nodes. Testing this, he found that it produced graphs that were much larger than the text files! So his final approach allowed only two successors to each node, with the use of dummy nodes (pointing to a null text string) to simulate more than two connections.

RJA/DEF/AIS considered the problem of editing versions of the archival file other than the most recent. If we have n versions, numbered 1 through n, but their history is a tree rather than a simple linear list, then it. may happen that the history of a particular record is not a single range of version numbers. If so, more space would be needed to store the version numbers in which a record is active, and there would be pathological cases in which this storage would grow unacceptably. They said that renumbering the versions in the tree could be used to solve this. However, the renumbering process takes time, so this presents another time/space tradeoff.

Another tradeoff, analyzed by KPB/TPM, arises from the choice of record size. With small records, the control information takes a significant fraction of the space in the archival file. With large records, however, there is the possibility that small changes cause large amounts of text to be duplicated. They recommended one- or two-line records for text files that were to be modified over an indefinite span of time.

DPH/LAH/YOM were interested in finding the minimal-space representation needed to represent the changes in an archival file. They did some research, suggested by DEK, into Kolmogorov complexity, and discovered that while there exists, for each pair of files, an optimally small "program" to describe the changes, and there exists a "universal decoder" program to interpret this encoding, there is provably no algorithm to find these optimal programs. However, there exists a computable function that will give arbitrarily tight upper bounds for the length of these programs.

Problem 3: 2 x 2 x 2 Rubicity

The object of this problem is to study a simplified version of Rubik's cube that is made out of eight cubies instead of 27. You can get the effect of a 2 x 2 x 2 from the ordinary 3 x 3 x 3 by painting all faces black except for the 24 faces adjacent to the eight **corners.**

This simplified puzzle can be rotated into at most $3^7 \cdot 7! = 11{,}022{,}480$ essentially different positions, because we can take one of the corner cubies (say the red-yellow-blue one) and consider it fixed; there are $7!$ permutations of the remaining corners and three orientations of each one.

Six "moves" are possible from any given position, namely to choose one of the three coordinate axes and to rotate a face perpendicular to it, either clockwise or counterclockwise, by 90 deg. The face that rotates is chosen as the face not containing the fixed red-yellow-blue cubie. Position x is said to be at distance k from position y if there is a way to get x to y in k moves but not in fewer than k.

Write a program that determines how many positions are at distance 0, 1, 2, . . . from any given position, and that finds two positions at maximal distance from each other. Your program should use at most about 200,000 computer words of 36 bits each; that's 7,200,000 bits, which is fewer than $3^7 \cdot 7!$. Try also to make your program fast.

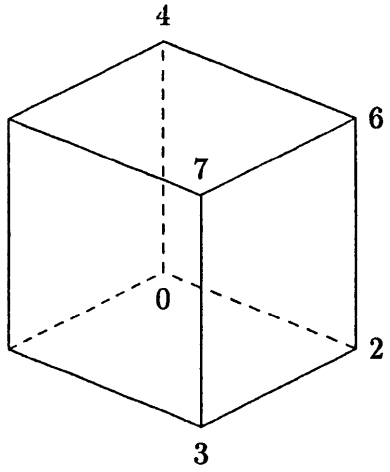
> ***If you can determine the key to unlock the knack for solving this puzzle,***
> ***the final triumph can be the psychological turning point in your life.***
> ***Mathematicians may be tested to the limit and cry over this one — and you may, too!***
> ***You will gain a measure of satisfaction, when you align one plane.***
> ***You will be delighted with the completion of two.***
> ***You will be elated with the completion of three or four!***
> ***The completion of the fifth plane will quicken your pulse!!***
> ***— and you will have scaled the peak once the last unit of the sixth plane falls into place!!!!***
>
> (Quoted from the box of a cube puzzle made in Taiwan)

Class notes for Tuesday, October 27

JSW started discussion of the third problem by showing a Rubik's cube that had all faces except the corners covered with tape-this could be used as a **2** x **2** x **2** cube. An interesting article about Rubik's cube, by Douglas Hofstadter, can be found in the April **1981** issue of Scientific **American.** It states that for the normal 3 x 3 x 3 cube, there are known positions whose distance is at least **17** moves (these include half-turns as well as quarter turns), and an upper bound on the maximum distance is conjectured to be either 22 or 23. (However, the best algorithm that has been devised for a human to solve the puzzle is only guaranteed to work in at most **52** moves.) This upper bound of 22 or 23 is also easily seen to be an upper bound for the **2** x **2** x **2 case.**

Next **JSW** asked whether the number of distinct positions was actually the number $3^7 \cdot 7!$, given in the problem description. Several people thought that they could reduce this figure somewhat, but no one had an immediate proof. To be able to discuss the problem more concretely, some time was spent on developing a notation that could describe configurations and moves of the cube. We need to name both the faces of the 8 cubies (as the sub-cubes are generally called), and the positions that they occupy. This is similar to the method used in solving the coin-flipping problem. AIS suggested that the three faces on one cubie be named, say, Green 0, Orange 0, and Yellow 0, and the faces on the cubie furthest away (which must contain the other three colors in the start position) be named Red 0, White 0, and Blue 0. Similarly there will be a face for each combination of a color and one of the numbers 0, **1,** 2, 3. JSW pointed out some problems with this, and suggested that, first of all, the cubies be given distinct numbers from 0 to 7, and also that instead of colors we can use a representation for the faces that is easier for a computer program to handle.

We can agree to let cubie 0 be the one that is always fixed; then there are many ways to assign the remaining numbers. **JSW** suggested the following labeling,

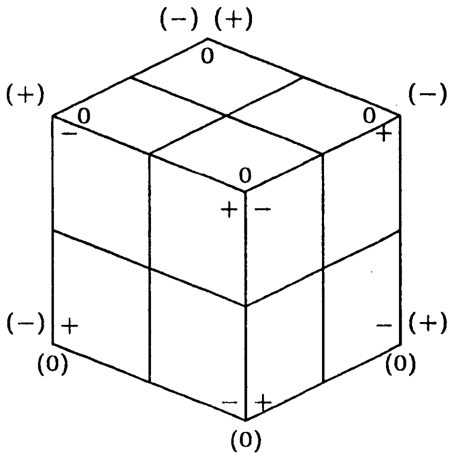


which has the property that the three directions away from the fixed cubic arc represented by the numbers 1, 2, and 4, and other cubies are labeled by the sum of the numbers describing their position. JCM noticed a parallel between this and Gray codes. A Gray code is an ordering of the numbers between 0 and 2” − 1, such that each number in the sequence, when written in binary, differs from the next by exactly one bit. For example,

$$000, 001, 011, 010, 110, 111, 101, 100$$

is a Gray code for $n = 3$. Such a sequence can be found for any n, and there is a "constructive" proof of this, that makes use of the sequence that has been found for $n - 1$ to construct a sequence for n. (If you look at the rightmost two bits in the sequence shown above, the pattern becomes clear.) KPB pointed out that JSW's numbering scheme has the property that each edge connects corners with numbers that differ in only one bit.

Once the cubies have been numbered, the faces can be labeled as in the following diagram (where parentheses enclose labels for the faces not visible).



The labels are chosen so that the top and bottom faces have O's, and so that on each corner cubie the faces are $\{0, +, -\}$ when read in counter-clockwise order. JSW also gave the names A, B, and C to the three clockwise,quarter-turns that leave cubie 0 fixed; these correspond to the permutations

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 5^- & 7^+ & 4^+ & 6^- \end{pmatrix},$$

$$B = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3^+ & 2 & 7^- & 4 & 1- & 6 & 5^+ \end{pmatrix},$$

$$c = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 6 & 2 & 4 & 5 & 7 & 3 \end{pmatrix},$$

where each one actually represents a permutation on all of the faces; the full permutation can be determined in a straightforward way from the information given. (This notation is very similar to what was used in the first problem.)

TPM conjectured that the fact that $\{0, +, -\}$ reads counterclockwise on each cubie forces a restriction on the set of possible configurations, and that $3^7 \cdot 7!$ can be reduced by a factor of 3. JSW asked if anyone could prove this, but no one was completely sure, so it was left to be discussed in the next class. DPH also asked whether it was possible to create a "circuit" of the cube that passes through each configuration exactly once; no one knew whether this could be done.

JF said that one way to arrive at the solution would be by an exhaustive search. She suggested using a table with one bit per possible configuration; then from a configuration whose distance is known, we can generate the positions that can be reached in one turn, mark them, and continue until all positions are marked.

Class notes for Thursday, October 29

Today we started by finishing the proof that the original number of positions, $3^7 \cdot 7!$, can be reduced to $3^6 \cdot 7!$, which is about 3.6 million. TPM stated that this is true because, using the notation of labeling faces $\{0, +, -\}$, we can prove that the sum of the faces on the top and bottom of the cube (counting **+1** for "+" and **-1** for "−") is always equal to 0 modulo 3. To prove this, we note that it is true in the original configuration, and that each of the quarter-turns A, B, and C (and their inverses) preserves this assertion. Therefore, it is always true after any sequence of quarter-turns.

This implies that once we have named the positions and orientations of six cubies (in addition to the one that is fixed from the start), the orientation of the seventh is not arbitrary; it can only have one possible value in order to satisfy the invariant just shown.

DMC suggested a lower bound on the maximum distance. Since there are only six moves from any position, and (except for the start position) one of those leads to a position at lesser distance, we can form a tree that has six branches at the first level and five at every other, and in which each node represents a position. The number of nodes must be at least $3^6 \cdot 7!$, since each position is represented by at least one node (some will have more than one), so if the depth of the tree is d, we must have

$$1 + 6 + 6 \cdot 5 + 6 \cdot 5^2 + \cdots + 6 \cdot 5^{d-1} \geq N,$$

or

$$d \geq \log_5 \frac{5-1}{6}(N-1) \approx 9.14,$$

where N = $3^6 \cdot 7!$. Therefore, the maximal distance is at least **10.** (This argument assumes that the number $3^6 \cdot 7!$ cannot be lowered further; the class did not prove this assumption.)

Now the discussion moved back to using an exhaustive search through all of the positions. With the reduction in the number of positions, we can now afford to use two bits per position and still stay in the main memory of our machines. (On a 36-bit machine such as the PDP-10, the exact requirement is 204,120 words of storage) In fact, one bit per position is not enough, since we may start with all positions at distance up to k

marked, but then positions at distance $k + 1$ have to be marked differently until we are completely done with a pass through all the configurations. JLM suggested a way that we could use a pair of bits to represent the necessary information: when we are updating the table for positions at distance $k + 1$, the four possible states could represent the properties
(1) Distance $< k$,
(2) Distance k,
(3) Distance $k + 1$,
(4) Distance not known (possibly $k + 1$, possibly greater).
As we move through. the table, whenever we see a position in class (2), we generate its "neighbors" and move those in class (4) to (3). Then when we have gone through the whole table, we make a second pass and move those from class (2) into (1), and (3) into (2), to prepare for the next pass. Everyone agreed that this is a reasonable way to do the exhaustive search.

However, it is not immediately clear how we can efficiently encode the configurations into the minimum space. JSW suggested that we consider first representing the $7!$ possible positions of the cubies, and then deal with the orientations. The problem is thus to put the permutations of $\{1,2,3,4,5,6,7\}$ in a one-to-one correspondence with the integers from 1 to $7!$. First, it was decided to work with the set $\{0,1,2,3,4,5,6\}$, and the integers from 0 to $7! - 1$. A simple indexing scheme is not sufficient, since any of the numbers could occupy any position, and the maximum code would thus be $7^7 - 1$. TPM came up with a method that works as follows: to encode a permutation, convert the original seven numbers to a different sequence of numbers, in which the nth one is always between 0 and $7 - n$. For example, the sequence $(6,0,1,5,4,3,2)$ would become $(6,0,0,3,2,1,0)$. The first number remains unchanged; after it is used, the remaining numbers are $\{0,1,2,3,4,5\}$ and 0 is number 0 in this set; then $\{1,2,3,4,5\}$ are left and 1 is number 0 in this set; then $\{2,3,4,5\}$ remain and 5 is number 3; and so on. Now, we use a mixed-radix notation to convert the new sequence into an integer, namely, we map $(a_0, a_1, \ldots, a_6)$ into the number

$$a_0 6! + a_1 5! \ldots + a_5 1!.$$

The number a_6 must always be 0, so it is left out of this expression. This presents us with a number between 0 and $7! - 1$ that is uniquely determined by the original permutation.

Now, to deal with the orientations, we note that these are six quantities that can each take on any of the values {0,+1, -1}. We can represent these in base-3 positional notation, by some mapping of $\{0, +1, -1\}$ onto $\{0,1,2\}$, to get a number between 0 and $3^6 - 1$. Call this number n_b, and let n_a be the number representing the permutation of the cubies. Then

$$n = n_b 7! + n_a$$

is a number between 0 and $3^6 \cdot 7! - 1$ that uniquely identifies the entire configuration of the cube.

There was some discussion of how we might get from the encoded value of a position to the encoded values of its neighbors without having to explicitly decode the configuration. No one was able to think of any such methods, but JSW suggested that even if this were impossible, the program might still run in a reasonable amount of time; therefore it was decided to deal with other issues first and come back to this later if it proved to be necessary.

Class notes for Tuesday, November 3

One topic discussed today was how to test the programs that were being written to solve the problem. JSW asked if anyone had come up with a simple test case that would see if their program ran without solving the main problem. When no one said they had, he suggested consdering the case of using only half-turns instead of quarter-turns of the $2 \times 2 \times 2$ cube. This should reduce the total number of configurations possible, but still can be solved with essentially the same method.

Some time was spent determining exactly how many configurations this new problem had. The first thing that was noticed is that orientations of all seven cubies are fixed. They start out in the "0" orientation, and half-turns never change this. Thus the factor 3^6 in the previous analysis is eliminated entirely, and there are thus no more than $7!$ states possible.

Next, it was pointed out that each cubie can occupy only some of the seven potential positions. Looking at the first diagram of the October 27 notes, it can easily be seen that positions $\{1, 2, 4, 7\}$ form a "closed set" under half-turns (or an ***orbit*** in the language of group theory), as do $\{3, 5, 6\}$. (Remember that cubie **0** is fixed.) This means that instead of $7!$ possible states, we now have at most $4! \cdot 3! = 144$.

The result is that limiting the moves to half-turns provides an extremely small test case for the programs.

Class notes for Thursday, November 5

Since the class finished last time with enough of the problem discussed to begin working on the programs, JSW decided to investigate the idea of reducing the space requirement by using equivalence classes of positions. As an example, suppose the sequence of turns $ABA^{-1}C^{-1}$ takes us to a position I? Then the position Q reached by the sequence $BCB^{-1}A^{-1}$ must be at the same distance from the start, since the only change is a permutation of the letters A, B, and C. In general, this will be the case whenever we make a permutation π of the six quarter-turn moves A, A^{-1}, B, B^{-1}, C, and C^{-1} which has the property that $\pi(X^{-1}) = \pi(X)^{-1}$ for each of the quarter-turns X.

Let us therefore call two positions P and Q equivalent if there exist sequences of quarter-turns σ_1 and σ_2 such that $P = \sigma_1(S)$ and Q = $\sigma_2(S)$, where S is the starting position, and $\sigma_1 = \pi(\sigma_2)$, for some permutation π of the type just described. It is easy to see that this satisfies the properties of an equivalence relation. If we work with the equivalence classes of positions under this relation, our algorithm will have to deal with significantly less than 3.6 million configurations. JSW said that the notation we have been using so far is clumsy for this approach, though.

AH described another way to use the idea of equivalence classes of positions. He claimed that if we rename the six colors on the faces of the cube, we can get to other positions that are at the same distance. There are 24 such renamings, because we can start the cube resting on any of the six faces, and then choose any of four faces to be the front, after which the other colors are fixed. DEF mentioned that we may be ignoring some other equivalences, which can be gotten by translating the cube across a major axis, instead of just rotating it.

KPB said he saw how we could get a reduction by a factor of six using JSW's method, or 24 using AH's equivalences. In fact, some further analysis revealed that we can increase this number. Using the definition of equivalence by legal permutations of the quarter-turns, the number of positions in each class depends on the number of different turns in the sequence used to get to them. If there is only one turn, then there are six possible permutations. If there are two (for example, $ABA^{-1}B$ uses the two letters A and B), then there are $6 \cdot 4 = 24$ equivalent sequences of turns. And if all three letters are used, there are $6 \cdot 4 \cdot 2 = 48$ sequences.

However, TPM saw that there must be something wrong with this, for if there is only one class with an odd number of members, the total number of positions cannot be even, which it is. SSK pointed out that some different renamings of quarter-turns might lead to the same positions. For example, the sequence AA is equivalent to $A^{-1}A^{-1}$, but this is not a different position. The class generated by AA has only three members, not six. JF asked whether we could ever get to a position at a different distance by such a permutation; TPM noted that this is impossible since if a position is at distance d, and a legal permutation of a sequence of moves that derives that position results in a position at distance $d' < d$, then applying the inverse of that permutation to a sequence of length d' to get to the second position would give us a sequence of length d' for the original position.

The object of all of this is to be able to use a bit table with only one or two bits per equivalence class. TPM mentioned that we have not yet determined how to decide what the classes are, and this is a significant part of the problem. KPB suggested that saving memory by a factor of 48 might free up enough space to store in a list structure the names of equivalence classes that we have found.

Turning to some different questions, DPH wondered what change is the distance a single move could perform. Certainly it cannot increase the distance by more than one, and neither can it decrease the distance by more than one. (Otherwise, we could get an increase of more than one in the other direction.) But can it cause the distance to stay the same? TPM proved that this cannot happen, because each of the quarter-turn moves is an odd permutation (a permutation is odd or even depending on whether it can be expressed as the product of an odd or even number of transpositions), and if two positions are at the same distance, then they both have either odd or even parity.

AH suggested that keeping track of the cube positions resulting from various move sequences might be easier than keeping track of the moves themselves. We can rename faces, if needed, during this checking. The main problem, still, seems to be finding a way to represent the equivalence classes. JSW suggested that we might look for one canonical member of each class. This would be a configuration that could be easily found as a function of any of the other members of its class.

This function would take as input the positions and orientations of the seven movable cubies in the configuration we are considering. JSW suggested that we first ignore the orientations, and just see what we can do with the positions. Suppose, for example, **that** we are given the sequence $(5,2,4,3,1,7,6)$ naming the cubies which occupy each of the seven positions. How do we get from this to one which is equivalent? AH said that with a lexicographic ordering on these 7-tuples, in which we would always let the first cubie be 1, for example, would result in a significant savings in storage.

JSW took a different approach, using the fact that cubic positions 1, 2, and 4 had been chosen to be "orthogonal." In the example, the cubies that occupy these positions are 5, 2, and 3, and we can rename the coordinates to put these in the order $2,3,5$, which is the smallest lexicoraphic ordering. The permutation of coordinates is given by

$$\begin{pmatrix}1\,2\,4\\4\,1\,2\end{pmatrix},$$

and we can complete this to a permutation on all seven cubie positions by viewing the permutation as the matrix

$$\begin{pmatrix}0&0&1\\1&0&0\\0&1&0\end{pmatrix}$$

operating on the binary number representing the position, where the number is represented by a column vector with the 4's place on top, the 2's place in the middle, and the 1's place on bottom. Thus cubic position 6 is mapped to position 3, since

$$\begin{pmatrix}0&0&1\\1&0&0\\0&1&0\end{pmatrix}\begin{pmatrix}1\\1\\0\end{pmatrix};$$

and cubie 7, which is in position 6 in the given configuration, is in position 3 in the canonical member of its class. Similarly, we compute the other positions, and get $(2,3,7,5,4,1,6)$ as the cubie positions in the canonical member.

Class notes for Tuesday, November 10

Having returned from his trip to Europe the past two weeks, DEK asked the class what progress they had made on the problem. RJA said that he was having trouble fitting the program into the main memory of the PDP-10. Even though the array of bits requires 204,120 words of 36 bits, and the 18-bit address space provides 262,144 words, there is not enough room to fit the program and the **SAIL** runtime library. DEK said that the same problem had come up in his METAFONT system, and provided a solution, using the "extended addressing mode" available on SCORE, which was programmed by David Fuchs. This mode allows a program to reference a 23-bit address space, although access to words with addresses higher than $2^18 - 1$ are more complicated than usual.

AIS, JF, and JCM, running their program on a VAX, had obtained a solution up to a distance of 10, but at that point the system had crashed. They said that over 1.4 million states were found with distance less than or equal to 10. They also ran the program using half-turns of the cube, and found a total of only 24 states, with a maximum distance of 4. If correct, this means that the analysis last week, resulting in at most 144 states, can be improved. JF mentioned seeing a paper by Joe Buhler from Reed College, which showed that the 2 x 2 x 2 cube had 11 levels, but this counted half-turns as single moves along with quarter-turns.

As a method for addressing the encoded states (the numbers in the range $[0, 3^6 \cdot 7! - 1])$, DEK suggested using "dope vectors." One could, for example, have a one-dimensional array Z, and given a permutaion (al,. . . , a_7), use the word $Z[Y[a_1, \ldots, a_6] + a_7]$ to store the encoded position. The function $Y[a_1, \ldots, a_6]$ could be computed similarly. JF wondered whether there was a way to transform a state to its neighbors without decoding its position at all. DEK said that he couldn't think of a way, although a clever encoding scheme might allow one of the six moves to be made easily.

AH proposed that by solving a problem similar to ours, namely considering the edges of the 3 x 3 x 3 cube and ignoring the corners, and combining this with the problem we were working on, we might be able to solve the general problem for the 3 x 3 x 3 cube. YOM noticed that this is a case of the composition of two processes, which was a useful notion in solving the coin-flipping problem. AH went on to say that he knew of sequences of moves that permute edges without affecting the corners—this could be used to combine the two problems. DEK now tried to compute the number of possible configurations for this problem. There are 12 edge cubies, each with two faces, so there are at most $12! \,.\, 2^12$ states. This probably reduces to $12! \,.\, 2^10$, he said, but that number is over 490 billion, which is far too large for our present methods.

DEK asked if there were other interesting variations of the problem. DMC suggested working with a 2 x 2 x 3 "cube." AH proposed using only two colors on faces. DEK's variation was to consider n-dimensional cub&. DPH mentioned a new puzzle that had recently appeared, called "Rubik's pyramid."

As a final point in the discussion of this problem, DEK said that he would like to see a good explanation of why there were only 24 states in the half-turn variation. He also said that the general problem of representing a permutation as a product of a given set of generators in the shortest possible way has been shown to be NP-hard.

Solutions for problem 3

Five groups presented solutions to this problem, and all produced the following data:

Distance from Starting state	Number of Positions
0	1
1	6
2	27
3	120
4	534
5	2256
6	8969
7	33058
8	114149
9	360508
10	930588
11	1350852
12	782536
13	90280
14	276

The maximum distance is **14,** and each group showed a position at this distance. The running times of the programs varied from 22 minutes (on a DECsystem-2060) to about 220 minutes (on a VAX-11/780).

All of the programs used the basic method outlined in class. Three of them (JEH/SSK, KPB/RJM, and DMC/LAH/YOM) saved some passes through the bit arrays by changing the meanings of the 2-bit codes on alternate passes. For example'on passes 1, 3, 5, etc. the code 10 might mean "seen this pass" and 01 might mean "seen last pass," while the reverse interpretation held on passes 2, 4, 6, etc. The advantage of this is that wc eliminate the step of changing "seen this pass" to "seen last pass" at the end of each pass by changing the meaning of the encoding, and we can convert "seen last pass" to "seen before last pass" during the time that we are checking all of the positions in the main loop.

Various methods were used to avoid expensive operations (multiplications, divisions, extracting bit fields from words) whenever possible. DMC/LAH/YOM used table lookups and one-dimensional arrays to a great extent. They also wrote the main loop of their program as stright-line code, with no sub-loops and no procedure calls, to save some overhead. On the other hand, KPB/RJM used **12** nested loops in their main procedure to avoid some encoding and decoding of position codes. JEH/SSK used a table of **18** different 2-bit masks, and a table of 2-bit state codes shifted into the proper position, so that testing the state of a 2-bit field in a word did not involve any shifting or division in the inner loop., AIS/JCM/JF realized that once all the positions found on the previous pass have been encountered, there is no need to scan the bit tables any more on the current pass, and implemented this simply by counting the number of such positions encountered.

A limited use of the equivalence of positions was made by RJA/DEF. Each time a new position was generated, they created six conjugates. These are formed from permutations of the cube about the O-7 axis and reflections across the O-2-5-7 plane. As discussed in class, these positions must all be at the same distance from the start. They also checked the set of conjugates thus generated for duplication of positions; that was used to avoid needless accesses to the bit arrays.

All of the programs run at SCORE (JEH/SSK, RJA/DEF, and KPB/RJM) used the extended addressing features. JEH/SSK had the fastest program, requiring only 22 minutes of CPU time.

Problem 4: Fractal Topography

An illustration that appears on the back cover of B. Mandelbrot's book *Fractals: Form, Chance, and Dimension* (Freeman, 1977) looks remarkably like a map of some familiar part of the earth's surface, maybe Australia, or Arabia, or . . . ; but on closer inspection we find that the map was actually generated by a random process and drawn by computer, following a curious procedure invented by Paul Lévy. The purpose of this problem is to generate some more examples of such maps, because Mandelbrot gives only one.

Here is how to construct the continents on a hypothetical planet called Stochastra:

a) Take a sphere and cover it with "Gaussian noise." This means that we generate independent, normally distributed random variables with mean zero and variance da for each infinitesimal surface area do of the sphere.

b) Define $h(p)$ for each point p on the sphere to be the integral of the Gaussian noise in (a), taken over the hemisphere centered at p.

c) Determine a threshold value ho so that the area of points p with h(p) $\geq h_0$ is $1/3$ the area of the sphere.

Then the land masses on Stochastra consist of the points with $h(p) \geq$ ho. (We can think of h(p) as the altitude at p, and h_0 is sea level.)

Write a program that draws a globe showing the continents on some randomly generated Stochastra. (Material that explains how to draw pictures using available hardware and software will be handed out.) The following theory should be helpful for getting started: If X_1 and X_2 are independent, normally distributed random variables with mean zero and respective variances v_1 and v_2, then $X_1 + X_2$ is a normally distributed random variable with mean zero and variance $v_1 + v_2$. It follows that the integral of the Gaussian noise in (a), taken over any region R of the sphere's surface, has the normal distribution with mean zero and variance equal to the area of R; furthermore the integrals over nonoverlapping regions are independent of each other. This leads to a neat property of the function constructed in (b): For all points p_1 and p_2 of the sphere, the value of $h(p_1) - h(p_2)$ is normally distributed with mean zero, and its variance is proportional to the great-circle distance between p_1 and p_2.

Class notes for Tuesday, November 10

DEK began the discussion of this problem by showing the class a copy of the cover of B. Mandelbrot's book *Fractals*. This provided an example of the kind of output that he wanted people's programs to produce. He defined the term "fractal" by saying this meant that if you look at a pattern though a magnifying glass, you still see the same structure.

When such a pattern contains a curve, and increasing the scale from s to 2s changes the length of that curve from L to L^x, then log, x is called the *dimension* of that curve. KYB asked whether dimension 2 is the limit of any curve constrained to a plane, and DEK answered that he didn't know, but the familiar space-filling curves do have dimension 2, and we will be working with curves of dimension $1\frac{1}{2}$.

Now DEK went over the description of the problem in more detail, and said that there is an analogy between it and the one-dimensional case of Brownian motion, In that domain, we have a "random" function $h(p)$, and the variance of $h(p_1) - h(p_2)$ depends on the distance between p_1 and p_2.

DEK began to review some of the basic concepts of normally distributed random variables. Suppose X_1 has mean x_1 and variance v_1, and X_2 has mean x_2 and variance v_2. If X_1 and X_2 are independent, it can be shown that X_1+X_2 has mean x_1+x_2 and variance $v_1 + v_2$. Let $X_3 = X_1 + X_2$.

Now, suppose we are given X_3 and X_2, and we want to find X_1. There is a way to "go backwards" and do this, which DEK said he would show in Thursday's class.

Class notes for Thursday, November 12

ARS had found a bug in the original statement of the problem, so DEK provided a correction. Recall $\mathrm{Var}(X) = E(x^2) - E(x)^2$. To construct the Stochastra:

a) Take a sphere and cover it with "Gaussian noise." This means that we generate independent, normally distributed random variables with mean zero and variance $\Delta\sigma$ for each infinitesimal surface area $\Delta\sigma$ of the sphere.
b) Define h(p) for each point p on the sphere to be the *sum* (not the integral) of the Gaussian noise in (a), taken over the hemisphere centered at p.

Equivalently,

a) Let the normally distributed random variables have mean zero and variance $1/\Delta\sigma$.
b) Define $h(p)$ by integrating the random variables.

The reason for this scaling change is that variances add. So

$$\mathrm{Var}(\sum X_i) = \sum \mathrm{Var}(X_i) = \sum \Delta\sigma = \int d\sigma$$

in the first case, and

$$\mathrm{Var}(\sum X_i\,\Delta\sigma) = \sum \mathrm{Var}(X_i\,\Delta\sigma) = \sum \mathrm{Var}(X_i)\Delta\sigma^2 = \sum \Delta\sigma = \int d\sigma$$

in the second case.

Now DEK described one-dimensional Brownian motion, which is a one-dimensional case of this problem. A Brownian function B from the real numbers to the real numbers has the property that for all $x < y$, the difference $U(y) - B(x)$ is a normally distributed random number with mean 0 and variance $y - x$. A normal *distribution* is the familiar bell-shaped curve, or more formally, X is a ***normal deviate*** (the word "deviate" means the same thing as "random variable") if

$$\Pr(X \leq x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-x^2/2} dx.$$

Note that a Brownian function is more than a random variable-it is a random function with the property that (B(y) − B(s)) $/\sqrt{y-x}$ is a normal deviate for all y > x.

Suppose X_1, X_2, and X_3 are independent normal deviates. Then $aX_1 + bX_2$ has the same distribution as $\sqrt{a^2+b^2}\,X_3$.

For Brownian motion, if $x_1 < x_2 < x_3$, then

$$B(x_2) - B(x_1) \text{ is normally distributed with variance } x_2 - x_1,$$
$$B(x_3) - B(x_2) \text{ is normally distributed with variance } x_3 - x_2,$$

and if we add these together,

$$B(x_3) - B(x_1) \text{ is normally distributed with variance } x_3 - x_1,$$

as it should be, because the variances add. The first one is $\sqrt{x_2 - x_1}\,X_1$ and the second is $\sqrt{x_3 - x_2}\,X_2$, where X_1 and X_2 are normal deviates, so following the rule we get $\sqrt{x_3 - x_2}\,X_3$.

KPB claimed that a function thus defined is not completely random. DEK agreed, but said that it has properties that make it a mathematically tractable object.

Before proceeding further with the study of Brownian motion, DEK suggested that we look at a discrete problem where we can "see" what is happening. Suppose we roll two dice; we know the probabability of the sum being each of the numbers $2, \ldots, 12$. For example, we will get 10 with probability $\frac{1}{12}$. Now, what if we are told that the sum is 10, and asked "what was the first number?" It must be 4, 5, or 6, and those are equally likely.

More generally, suppose that $p_0, p_1, p_2, \ldots$ are the probabilities that the first roll is $0, 1, 2, \ldots$, and $q_0, q_1, q_2, \ldots$ are the probabilities that the second roll is $0, 1, 2, \ldots$. Then the probability that the sum is 10 is

$$\Pr(X_1 + X_2 = 10) = p_0q_{10} + p_1q_9 + \cdots + p_{10}q_0.$$

We could generate a random number with this distribution in some way other than rolling two dice. Now, what is the probability that the first roll is 3? It is

$$\Pr(X_1 = 3 \mid X_1 + X_2 = 10) = \frac{p_3q_7}{p_0q_{10} + \cdots + p_{10}q_0}.$$

By such considerations it is possible to generate a random variable X_1 *after* generating the random quantity X_1 + X_2, in such a way that the results obtained are equivalent to what we would have if X_1 had been generated first.

Now let's turn to the Brownian function, in which the random variables are *normal deviates* X, where

$$\Pr(x \le X \le x + \Delta x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \Delta x + \text{low order terms.}$$

For normal deviates, it can be shown that the distribution of pairs $(\frac{1}{2}X_1 + \frac{1}{2}X_2, X_1)$ is the same as the distribution of pairs $(\frac{1}{\sqrt{2}}X_3, \frac{1}{\sqrt{2}}X_3 + \frac{1}{\sqrt{2}}X_4)$. We can use this to compute a Brownian function at two points $B(\frac{1}{2})$ and $B(1)$ in two equivalent ways: If we choose $B(x)$ by having $B(0) = 0$, $B(\frac{1}{2}) = \frac{1}{\sqrt{2}}X_3$, and $B(1) = \frac{1}{\sqrt{2}}X_3 + \frac{1}{\sqrt{2}}X_4$, we will get the same distribution of values at $\frac{1}{2}$ and 1 as if we computed $B(0) = 0$, $B(1) = X_1$, and $B(\frac{1}{2}) = \frac{1}{2}X_1 + \frac{1}{2}X_2$.

To prove the claim in the previous paragraph, notice that in the first case

$$\Pr(\tfrac{1}{2}X_1 + \tfrac{1}{2}X_2 = x \text{ and } X_1 = y) = \Pr(X_1 = y \text{ and } X_2 = 2x - y).$$

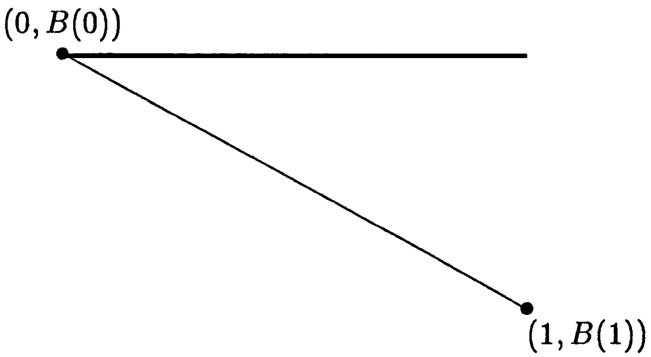
So the density is proportional to $e^{-\frac{1}{2}y^2} e^{-\frac{1}{2}(2x-y)^2}$. (The constant of proportionality doesn't matter because it can be obtained by integrating over all x and y.) In the other case, we have

$$\Pr(\tfrac{1}{\sqrt{2}}X_3 = x \text{ and } \tfrac{1}{\sqrt{2}}(X_3 + X_4) = y),$$

which works out to have a density proportional to $e^{-\frac{1}{2}(\sqrt{2}x)^2} e^{-\frac{1}{2}(\sqrt{2}y - \sqrt{2}x)^2}$. Both of these simplify to the expression

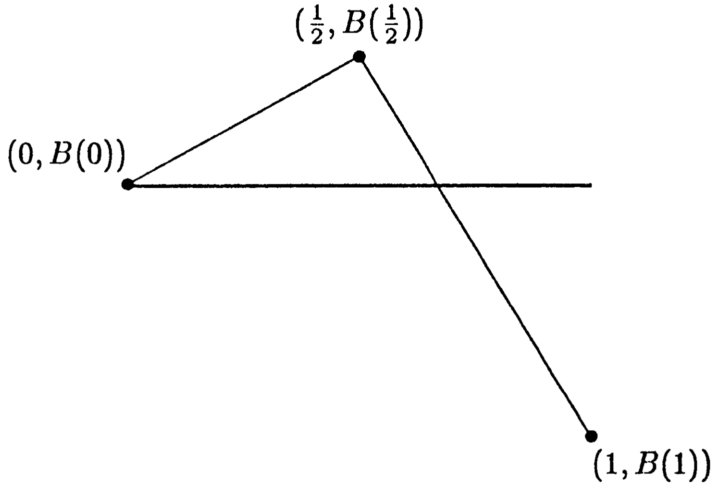
$$e^{-\frac{1}{2}(4x^2 - 4xy + 2y^2)}.$$

Suppose we want to compute a Brownian function B at 1024 points. Without loss of generality, let $B(0) = 0$. We could divide $[0, 1]$ into 1024 intervals, and at each one, take a normal deviate and multiply by $\sqrt{1/1024}$. But we may not want to do all this work. So, DEK showed an example of a faster calculation. We choose $B(1)$ at random. (In his example, $B(1) = -.54$.) Now, we imagine a line drawn as follows.



To find $B(\frac{1}{2})$, we take $\frac{1}{2}(-.54) + \frac{1}{2}X$, where X is another normal deviate. In the example, this resulted in $B(\frac{1}{2}) = +.23$. Thus, B changes by $+.23$ going from 0 to $\frac{1}{2}$, and by $-.77$

going from $\frac{1}{2}$ to 1.



In general, when we split an interval of length l in half, and the change in B over this interval is v, we choose a normal deviate X, and make B change by $\frac{1}{2}v + \frac{1}{2}\sqrt{l}X$ in the first half, and $\frac{1}{2}v - \frac{1}{2}\sqrt{l}X$ in the second half. Thus, a summary of the computation is

$$\begin{aligned} B(1) &= X_0, \\ B(\tfrac{1}{2}) &= \frac{B(0)+B(1)}{2} + \frac{1}{2}X_1, \\ B(\tfrac{1}{4}) &= \frac{B(0)+B(\frac{1}{2})}{2} + \frac{1}{\sqrt{8}}X_2, \\ B(\tfrac{3}{4}) &= \frac{B(\frac{1}{2})+B(1)}{2} + \frac{1}{\sqrt{8}}X_3, \end{aligned}$$

etc.

From this tableau it is clear that the Brownian curve is self-similar; i.e., small pieces look the same as large ones when magnified. For example, if we look only at the values of *B(x)* for x between 0 and $\frac{1}{2}$ we see that they are obtained by carrying out the same pattern that was used for x between 0 and 1, but multiplied by $1/\sqrt{2}$. Thus $B(2x)/\sqrt{2}$ is also a Brownian function.

The formulas can be "unwound" to obtain a general formula for $B(r)$ when $0 \le r < 1$ and r is any *dyadic* rational, i.e., a fraction whose denominator is a power of 2:

$$\begin{aligned} B(r) = rX_0 + \|r\|X_1 + 2^{-1/2}\|2r\|X_{\lfloor 2(r+1)\rfloor} \\ + 4^{-1/2}\|4r\|X_{\lfloor 4(r+1)\rfloor} \; 8^{-1/2}\|8r\|X_{\lfloor 8(r+1)\rfloor} \; \ldots, \end{aligned}$$

where $\|x\|$ denotes the distance from x to the nearest integer (hence $\|x\| \le \frac{1}{2}$).

When r is a dyadic rational, this infinite series is actually finite, because $\|2^n r\|$ will be zero for all large n. When r is not a dyatlic rational, we can use this infinite series to define *B(r),* as a function of the infinitely many normal deviates $X_0, X_1, X_2, \ldots$.

Since normal deviates are not bounded, it is possible for the infinite series to diverge; but this has probability zero. In order to get a feeling for what is going on, let's prove in fact that it is extremely likely that the series

$$X_1 + 2^{-1/2}X_{\lfloor 2(r+1)\rfloor} + 4^{-1/2}X_{\lfloor 4(r+1)\rfloor} + \cdots$$

is less than 24, for all $0 \le r < 1$: Let $c = 6$ and $\alpha = .75$. If $X_1 < c$ and $2^{-1/2}X_2 < \alpha c$ and $2^{-1/2}X_3 < \alpha c$ and $4^{-1/2}X_4 < \alpha^2 c$ and $\cdots$ and $2^{-\lfloor \lg k\rfloor/2}X_k < \alpha^{\lfloor \lg k\rfloor}c$ and $\ldots$, the stated series is less than $c + \alpha x + \alpha^2 c + \ldots = c/(1-\alpha) = 4c = 24$, as desired. On the other hand, the probability that these inequalities are not all satisfied is

$$\begin{aligned}\Pr\Big(X_1 \ge c \text{ or } \tfrac{X_2}{\sqrt2} \ge \alpha c \text{ or } \tfrac{X_3}{\sqrt2} \ge \alpha c \text{ or } \tfrac{X_4}{\sqrt4} \ge \alpha^2 c \text{ or } \cdots&\\ \text{or } \tfrac{X_k}{\sqrt2^{\lfloor \lg k\rfloor}} \ge \alpha^{\lfloor \lg k\rfloor}c \text{ or } \cdots\Big)&\\ \le \Pr(X_1 \ge c) + \Pr(X_2/\sqrt2 \ge \alpha c) + \ldots&\\ \le T(c) + 2T(\sqrt2\,\alpha c) + 4T((\sqrt2\,\alpha)^2 c) + \cdots&\end{aligned}$$

where T(x) is an upper bound on the probability that a normal deviate is x or more. We have

$$\begin{aligned}\Pr(X \ge x) &= \frac{1}{\sqrt{2\pi}}\int_x^{\infty} e^{-t^2/2}\,dt\\ &< \frac{1}{\sqrt{2\pi}}\int_x^{\infty} e^{-t^2/2}\frac{t}{x}\,dt = \frac{1}{\sqrt{2\pi}\,x}e^{-x^2/2},\end{aligned}$$

so we can set $T(x) = e^{-x^2/2}/(\sqrt{2\pi}\,x)$. Letting

$$\begin{aligned}Q(\alpha,c) &= \sum_{k\ge0} 2^k T\big((\sqrt2\,\alpha)^k c\big)\\ &= \sum_{k\ge0} \frac{1}{\sqrt{2\pi}}\frac{2^k}{(\sqrt2\,\alpha)^k c}e^{-\frac12((\sqrt2\,\alpha)^k c)^2},\end{aligned}$$

and using MACSYMA, one can compute

$$\begin{aligned}Q(.75,6) &= 1.2139033 \times 10^{-9} \text{ after 1 term,}\\ &= 1.24409 \times 10^{-9} \text{ after 2 terms,}\\ &= 1.247396 \times 10^{-9} \text{ after 3 terms,}\\ &= 1.2476621 \times 10^{-9} \text{ after 4 terms,}\\ &\approx 1.25 \times 10^{-9}.\end{aligned}$$

This tiny number is a generous upper bound on the probability that the stated infinite series could exceed 24 at any point r.

DEK had written a SAIL program to solve a one-dimensional problem similar to problem 4. Suppose we are given a Brownian function, and want to know where it crosses $\frac{2}{3}$. There might be many such places, since near a crossing, the function "wiggles" up and down. The program reports all cases in which $B(l) < \frac{2}{3} <$ B(r), where l and r are dyadic rationals and $|l - r|$ is less than some threshhold 6.

The program works as follows. Let M be the larger of the values $B(I)$, $B(r)$; and let m be the smaller. If $M + \sqrt{r-l} \cdot 12 < \frac{2}{3}$ and m $- \sqrt{r-l} \, . \, \mathbf{12} > \frac{2}{3}$, the program does not look between l and r for a crossing, because the probability of finding one is very small. Otherwise, it takes the midpoint between l and r and looks at the two subintervals thus formed. Whenever $m < \frac{2}{3} < M$, the program has found a solution.

ARS mentioned that a reflection principle could make this more efficient. He gave a brief description of what this meant. (See the notes for the November 17 class.)

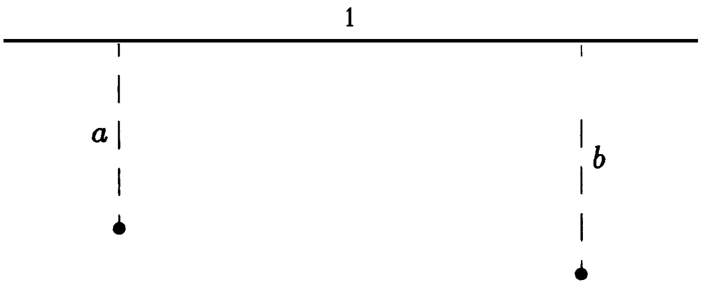
Now DEK explained how this could be applied to the Stochastra problem. We divide the surface of the sphere into some number of regions. Each region has a value, which is a normal deviate. Now, if we decide to look for a crossing of sea level in a region, we split it into two parts, and need to get two numbers that add to the previous value. To do this, we use the fact that the distributions of

$$(aX_3, aX_3 + bX_4) \quad \text{and} \quad \left(\frac{a^2X_1 + abX_2}{\sqrt{a^2+b^2}}, \sqrt{a^2+b^2}\,X_1\right)$$

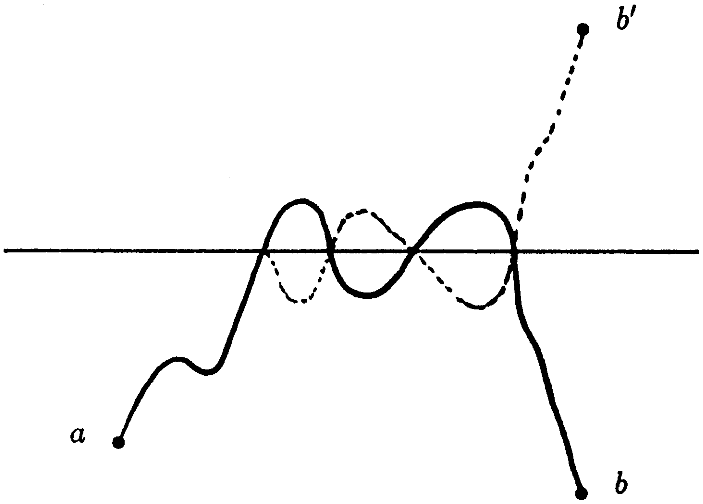
are the same. (The numbers a and b come from the ratio of the areas.)

Class notes for Tuesday, November 17

Using the reflection principle suggested by ARS, DEK said that he was able to improve the program that he discussed last time. If we are given two points on the same side of a line, we would like to find out the probability that a Brownian function through these points crosses the line between them. If they are at distances a and 6 from the line, and their horizontal separation is **1**,



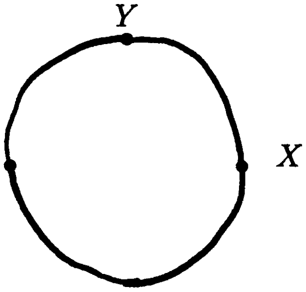
it turns out that this probability is exactly e^{-2ab}. To prove this, suppose that the curve crosses the line. Then construct a new curve that is the reflection of the old one, after the first point at which it crosses.



This new curve is as likely to occur as the original one, because it can be obtained from the other by negating normal deviates, and normal deviates are symmetric about zero. By this reasoning, the probability of a curve from a to b crossing the line is the ratio of the number of curves from a to b' to the number of curves from a to b, namely $e^{-(a+b)^2/2}/e^{-(a-b)^2/2} = e^{-2ab}$. There might be something similar in the two-dimensional case, and DEK said that this would be a good research topic.

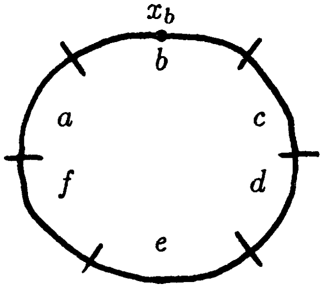
If we are working with an area on a sphere, RJM asked, and want to find out if there exists a zone within which a random function crosses zero, why don't we do a reflection as in the one-dimensional case? DEK answered that we aren't computing a Brownian function in the same way, since it is defined by a sum. The one-dimensional version of that would be to compute the function at each point by summing a random function in an interval around the point.

Using a circle rather than a line segment gives a one-dimensional analog to the Stochastra problem. JCM suggested looking at a change in the coordinate system.



Rather than summing in the semicircle surrounding Y, we get the same result by summing in the semicircle preceding X. DEK said that the Brownian motion example was like summing over all the area to the left of a point; $B(y) - B(x)$ is a random variable. KPB remarked that by summing over a hemisphere, we get a smoothing effect. DEK added that this causes "mountains" in the terrain to look like real mountains.

DPH suggested a way to attack the problem on a circle. First we divide the circle into some number of segments, and generate a "noise" value for each segment. Then, at the midpoint of each segment, we add the noise over the semicircle around that point.



For example, we have $f(x_b) = a+b+c$. As we move from x_b to x_c, we add d and subtract a. DEK said that instead of generating all the values at the beginning, we can first assign a random value to a + b + c, and then choose another random number for $d -$ a, since it is the sum of two random deviates.

LAH noticed that the Stochastra problem has the following property: If you can see half of the sphere, then the other half is determined. Or, as DEK put it, f(p) + $f(p')$ is a constant independent of p, where p' is the antipodal point of p. LAH went on to say that in the one-dimensional case, we can let

⌧☎♦◑ ▪ ☒☎♦◑ — ♦ . x(1).

This function (which probability theorists call a "Brownian bridge") doesn't have the antipodal property. He asked if this could be generalized to a sphere. DEK doubted this, saying that Lévy's definition is the only one that has been proved to "work well."

The discussion now turned to searching for ways to divide the surface of a sphere is a "regular" way. SSK suggested inscribing a regular tetrahedron inside the sphere, then drawing segments of a great circle through each pair of points. This gives four regions of equal area, and we can assign them random values. Now, we can subdivide one of the regions by taking a face of the tetrahedron, dividing it into four equilateral triangles, and making great-circle lines. DEK reminded the class that the area of a spherical triangle, given the angles A, B, and C of the triangle, is equal to $A + B + C - \pi$, on a sphere of radius 1. This can be generalized to show that the area of an n-gon on the sphere is $A, + . \cdot\cdot + A, - (n-2)\pi$.

RJM thought that it could be shown that the four triangles in SSK's construction have equal area, but SSK proved that they do not. He said to consider the solid angles subtended by each of the triangles. They must be different, so the triangles have different area. JCM claimed that if you know the constant of proportionality between these angles, you can still make use of the construction.

DPH suggested beginning by inscribing a cube in the sphere, but many people noticed that there is no regular way to continue this construction after the first step. JF asked if the reason for doing this was to get the self-symmetry property.

Abandoning the idea of a completely regular way of dividing the sphere, KPB suggested using longitude and latitude coordinates, and compensating for the irregular areas by using a weighting factor that would increase near the poles.

A few more ideas were brought up as the class neared the end of its time. DPH asked about DEK's use of the reflection principle earlier—once we have points a and b, and we decide by choosing a random number that the curve does cross the line between them, how do we proceed to find the crossing? DEK answered that once we know there is a crossing, we use the point b' instead of b, and change the problem into finding a random path from a to b'.

RJM noticed that in the one-dimensional case we start with an interval, and use one random number each time we divide it in half. In the two-dimensional case, we split an area into four sections, so we need to deal with three random numbers at once. DEK said that we could avoid this by using triangles, and bisecting them at each step, but SSK argued that this would make information hard to keep track of.

Another idea that was proposed was first to create a sphere covered with a random noise function G_1, and then to define a new noise function by

$$G_2(x) = \begin{cases} 0 & \text{in the east hemisphere,} \\ G_1(x) - G_1(\bar{x}) & \text{in the west hemisphere,} \end{cases}$$

where $\bar{x}$ is the point antipodal to x. The sum over the hemisphere about a point x is then the same, except for a constant factor:

$$\int_{t\in h(x)} G_2(t) = \int_{t\in h(x)\cap W} (G_1(t) - G_1(\bar{t})) = \int_{t\in h(x)} G_1(t) - \int_{t\in E} G_1(t),$$

where h(x) is the hemisphere about x, since $t \in h(x) \cap W$ if and only if $\bar{t}$ is in E but not in h(x).

Class notes for Thursday, November 19

DEK began the class by asking what coordinates people thought should be used to represent points on a sphere. JF suggested spherical coordinates, in which the radius would be fixed, and θ (the longitude) and ϕ (the latitude) would be specified. DEK reminded everyone that we can translate between this and Cartesian coordinates by means of the formulas

$$x = \sin\phi\cos\theta,$$
$$y = \sin\phi\sin\theta,$$
$$z = \cos\phi,$$

and that these coordinates satisfied the relation $x^2 + y^2 + z^2 = 1$. We can transform the coordinates of a point by matrix multiplication,

$$U\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} u_{11}x + u_{12}y + u_{13}z \\ u_{21}x + u_{22}y + u_{23}z \\ u_{31}x + u_{32}y + u_{33}z \end{pmatrix}.$$

If U happens to be a unitary matrix (this means that $U^{-1} = U^T$, or, equivalently, that $U^TU = I$), then

$$(UX)^T(UX) = X^T(U^TU)X = X^TX,$$

so the dot product of a vector with itself (which is its length) is preserved when it is multiplied by a unitary matrix.

RJM asked how we were going to determine what altitude would be the sea level. Since we are dealing with an infinite number of points, we can use some analysis prior to any computation to do this. Recalling that sea level was to be chosen so that $\frac{2}{3}$ of the globe is water and $\frac{1}{3}$ is land, DEK asked whether there would necessarily be a single hemisphere that had this property. DPH answered that there would be such a hemisphere. DEK began to answer RJM's question by saying that a crude approximation could be obtained by using a small number of points. RJM noted that since the heights at antipodal points are opposite, we can rotate the sphere to get less or more area above sea level.

Now let's assume, said DEK, that we can't determine the $\frac{2}{3}$ level in advance (since he didn't see any way to do so). DPH claimed that if we can find an expected value for the $\frac{2}{3}$ level, we can create a map on the sphere and then rotate if necessary it to get a hemisphere with $\frac{2}{3}$ water. YM thought that this would be a good approximation. SSK said that the variance of this method would decrease as the number of points used increases.

Since the Gaussian noise function is a random, its integral given by the function h is random, and the measure of the set $\{p : h(p) \geq s\}$is a random variable. We want this to have expected value $\frac{2}{3}$. YM remarked that the integral of a Gaussian noise function is the *error function*

$$\mathrm{erf}(x) = \int_{-x}^{x} \frac{e^{-t^2}\,dt}{\sqrt{\pi}}.$$

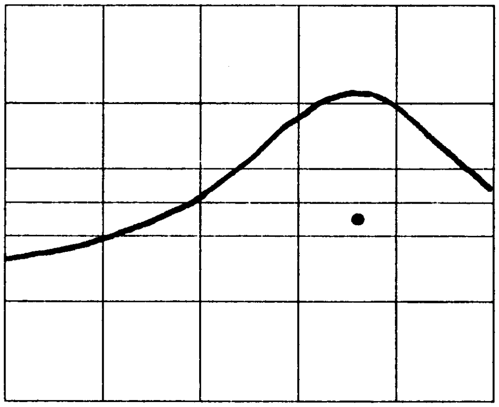
KPB said that we may want to insert some human intelligence halfway through the process. We could have the computer generate some samples, and then look and see what is best. DEK said that some people may also want to compute s exactly by a method that converges to it.

SSK said that if we break up the globe into pieces that get thinner, and then split those into new pieces, this two-step process can be used to compute the areas of the new pieces. AH said that all the pieces could have the same area, but different shapes. Using spherical coordinates, for example, we could have equal increments of the longitude θ, but vary the increment of the latitude ϕ (more near the poles, less near the equator) so that the areas are the same. DMC said the formulas should be fairly simple, and DEK provided the fact that the area of a ring on a sphere, defined by two latitudes, is proportional to the distance between the planes containing the top and bottom of that ring. So we can divide the height of the sphere by m and project this onto the appropriate latitudes, and create n equally-spaced longitude lines, to get mn equal areas.

Now, DPH said, the problem was to minimize the error gotten by adding or omitting pieces. The pieces in question are those at the edge of hemisphere, DEF added, since an arbitrarily drawn hemisphere would intersect only a part of some of the areas. DEK asked how much this error could be. If we compute h by adding the values at all the pieces a hemisphere touches, then the error is at most the number of sections at the edge of a hemisphere, DEF answered. TPM said this is roughly the smallest width of any piece, times the length of a. semicircle. DPH noticed that there are at most m + n such pieces, which can be seen by looking at a planar representation of the coordinate system and the path of a great circle. DEK remarked that as we generate a value for each of the mn pieces, we can add it to the accumulated values for each of the points in its hemisphere.

Now DEK changed the topic by saying that when we try to solve a problem such as this one, it is sometimes not best to choose points totally at random, but to have a method with some regularity in it. An example he gave is the solution of differential equations. SSK asked how we would choose random points. DEK suggested first choosing the value of z as a random number in the interval $[-1, 1]$. Then choose a random point on a circle, and multiply its coordinates by $\sqrt{1-z^2}$. A random point on a circle can be gotten by choosing random pairs (x, y) until $x^2 + y^2 \leq 1$ and then using the point $(x/\sqrt{x^2+y^2}, y/\sqrt{x^2+y^2})$. KPB said that we could, instead, choose an angle θ at random. DPH remarked that this method is equivalent to choosing integers in the ranges $[1,m]$ and $[1,n]$ at random, and using them as indices to select, one of the pieces that we divided the sphere into.

A few more ideas came up at the end of the class. RJM said that it seemed that we could take two points, and find out if there is a crossing of sea level between them, as in the Brownian motion problem. DEK said that a program could maintain a data structure to represent a division of the sphere into polygons, with links to point to adjacent regions. When we cut a polygon with a great circle, we update this data structure. He then made a sketch of what a hemisphere about a point would look like using the m-n model. It is easy for a point on the equator. If we move up or down, we get a picture such as the following.



The error in the computation will be about the square root of the number of small pieces. Converting to a point on a plane, by projecting, should be fairly easy, he claimed. JF suggested using a stereographic projection.

Class notes for Tuesday, November 24

DEK asked if anyone had general comments about the problem. AIS mentioned that someone he knew had generated surfaces in a raster graphics system. The amount of computation needed was only that required to accomodate the resolution of the particular raster size it was working with. This method generated interesting drawings of planets.

AH said that our model doesn't reflect everything we know about the earth. For example, continental drift and the rotation of the planet are not accounted for. DEK said that Mandelbrot had noticed the dimension of the coastline was 1.5 in the model, but actually only 1.3 on the earth, so he devised a more complicated model (which does not appear in the book). **AH** added that on the earth there are also more flat areas than our model will produce.

DEK asked the class how far they had progressed in their solutions. KPB said that his group had devised an algorithm, but not written any code yet. Given the coordinates (x_0, y_0, z_0) of a point p, we can find the boundary of its hemisphere as follows. First we convert the position into spherical coordinates (0, ϕ), always using a sphere of radius **1.** Recall that

$$\begin{aligned} x &= \sqrt{1-z^2}\cos\theta = \cos\phi\cos\theta, \\ y &= \sqrt{1-z^2}\sin\theta = \cos\phi\sin\theta, \\ z &= \sin\phi. \end{aligned}$$

Now we rotate the point to the north pole using a unitary matrix. **We** want U such that

$$U\begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

First, DEK observed, if we use

$$U_1 = \begin{pmatrix} \cos\theta' & \sin\theta' & 0 \\ -\sin\theta' & \cos\theta' & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

then

$$U_1\begin{pmatrix} \cos\phi\cos\theta \\ \cos\phi\sin\theta \\ \sin\phi \end{pmatrix} = \begin{pmatrix} \cos\phi\cos(\theta-\theta') \\ \cos\phi\sin(\theta-\theta') \\ \sin\phi \end{pmatrix}.$$

So U_1 changes the longitude of p by the angle θ', and we can transform p into a point with $\theta = 0$. Now, we need a matrix U such that

$$U\begin{pmatrix} \cos\phi \\ 0 \\ \sin\phi \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

If we use

$$U_2 = \begin{pmatrix} -\sin\phi' & 0 & \cos\phi' \\ 0 & 1 & 0 \\ \cos\phi' & 0 & \sin\phi' \end{pmatrix},$$

we get

$$U_2 \begin{pmatrix} \cos\phi \\ 0 \\ \sin\phi \end{pmatrix} = \begin{pmatrix} \sin(\phi-\phi') \\ 0 \\ \cos(\phi-\phi') \end{pmatrix}.$$

Therefore we can choose ϕ' to make $\phi - \phi' = 0$. So the final matrix we want is

$$U = U_2U_1 = \begin{pmatrix} \sin\phi\cos\theta & -\sin\phi\sin\theta & \cos\phi \\ -\sin 0 & \cos\theta & 0 \\ \cos\phi\cos\theta & \cos\phi\sin\theta & \sin\phi \end{pmatrix}.$$

Since U is unitary, we can use $U^{-1} = U^T$ to convert back, if we want.

KPB said that he had done things a bit differently. Letting ϕ_0 be the angle between p and the equator. If p is at the north pole, its hemisphere is bounded by the circle $z = 0$. Otherwise, we use a coordinate transformation to convert the equation $z' = 0$ (in the coordinate system where p is the north pole) to

$$z = \frac{\tan\theta\cos\phi_0}{\sqrt{\tan^2\theta + \sin^2\phi_0}},$$

which can be rewritten as

$$z = \frac{\tan\theta\sqrt{1 - z_0^2}}{\sqrt{\tan^2\theta + z_0^2}}.$$

There is a problem, though, as $\tan\theta$ approaches ∞, since this will cause the sign of z to change from positive to negative abruptly. Therefore we should use the above formula to compute $|z|$, and use a bit of "intelligence" to get the correct sign. DEF said that if we multiply both the numerator and denominator by $\cos\theta$, we get

$$z = \frac{\sin\theta\cos\phi_0}{\sqrt{\sin^2\theta + \sin^2\theta_0\cos^2\theta}} = \frac{\sin\theta\sqrt{1 - z_0^2}}{\sqrt{\sin^2\theta + z_0^2\cos^2\theta}},$$

and no longer have the problem. KPB also added that his formula assumes $\theta_0 = 0$; adding appropriate terms involving $\theta - \theta_0$ will correct for this.

AH now said that a point (x_1, y_1, z_1) is in the hemisphere around another point (x_2, y_2, z_2) if and only if $x_1x_2 + y_1y_2 + z_1z_2 \geq 0$. DEK interpreted this as saying that the cosine of the angle between the points is non-negative. TPM guessed that after transforming to spherical coordinates, this might yield the same equation as he and KPB had derived.

JCM and AIS had gotten a program running, and brought along some output to show the class. All of their hemispheres had a large land mass concentrated in the center; this seemed to indicate a bug somewhere. Their method did not involve trying to draw a curve at the land/ocean boundary; instead they drew lines scanning across the globe, and changes from white to black, or vice versa, whenever they crossed the shoreline. RJM also had some output to show from his program.

DEK commented on debugging programs that work with random numbers. Since different things may happen on each test run, bugs become harder to find. Along the same lines, he gave the example of chess-playing programs, which may have hidden bugs that are reflected in play that is poor, though still legal. DMC added that some programs contain code that is never executed in normal circumstances.

RJM said that one of his tests was to print out the variance with each drawing, and to check it by hand. He asked if there were any debugging techniques that would be appropriate. DEK mentioned the following: (1) be able to control the seed of the random number generator, so that it can be duplicated when a strange bug occurs; (2) put in redundant code to check for things that shouldn't happen; (3) look at the output for "random" conditions that seem to be happening too frequently; (4) in general, use statistical inferences.

Solutions for problem 4

The programs for this problem were all written in either SAIL or C. Those using the SAIL computer system had access to display routines and could produce sample output on the Xerox Grapics Printer (XGP).

In general, the problem was divided into the generation of the map of Stochastra, **and** its display. Those groups not using SAIL were able to produce output on SUN terminals or the Dover printer.

JMC/AIS used the basic approach presented in class. They divided the sphere into an m-by-n grid, with each portion having the same area, and assigned a Gaussian noise value to each entry. Heights were computed by considering each latitude in turn. For each latitude a point was chosen and all the values in its hemisphere were summed; then by shifting the hemisphere, heights for the other points with the same latitute were computed.

To compute the sea level, each point's height was used to increment a counter corresponding to a range of values that it occurred in (called a "bucket"), and then the numbers of entries in the buckets were used to determine the cutoff which would make $\frac{1}{3}$ of the **area** be land.

KPB's approach was similar. He saved a factor of 2 using the fact that heights of antipodal points must sum to zero, so only one hemisphere's heights need to be computed. (This fact was also used by some other groups.) In KPB's program, the value used to determine sea level is left to be selected in the graphics output phase.

JF/DPH/TPM used a binary search to determine the location of sea level, as did some of the other groups.

RJM began by implementing the straightforward "brute force" method, but found that it was too slow, so he added the improvements discussed in class. He also found that table lookup instead of calls to library routines for trigonometric functions sped up his program.

The display routines had a number of useful features. JEH/DEF/SSK implemented commands to raise or lower the sea level, to display the globe with a constant z and a varying θ, or vice versa, and to output drawings on the XGP. KPB also had commands to read pictures from different files, as produced by the generation program, and display them at varying resolutions.

JCM/AIS used standard Unix library routines to handle much of the work of plotting and sending output to the Dover, and their drawing program produced output in the form required by these programs.

DMC considered two ways of producing the display. Given that the globe is divided into "rectangular" pieces, his first approach considered each pixel to be displayed and darkened it if the corresponding rectangle was above water. This turned out to be too slow. The better method was to look at each piece of the sphere, and darken or leave blank all of the pixels corresponding to that rectangle.

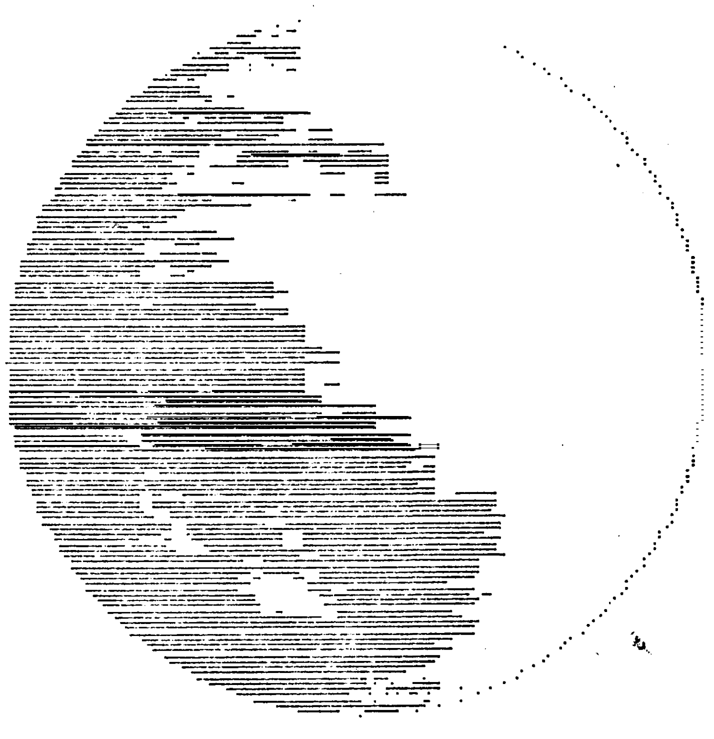
LAH/ARK/YOM implemented an approach quite different from that discussed in class. First, they analyzed the one-dimensional analog of the problem and found that the solution in class requires space that is worst-case $O(\log_2 n)$, where n is the number of points being displayed, and asymptotically achieves this worst case with probability 1. They gave an algorithm to solve the problem in constant space instead.

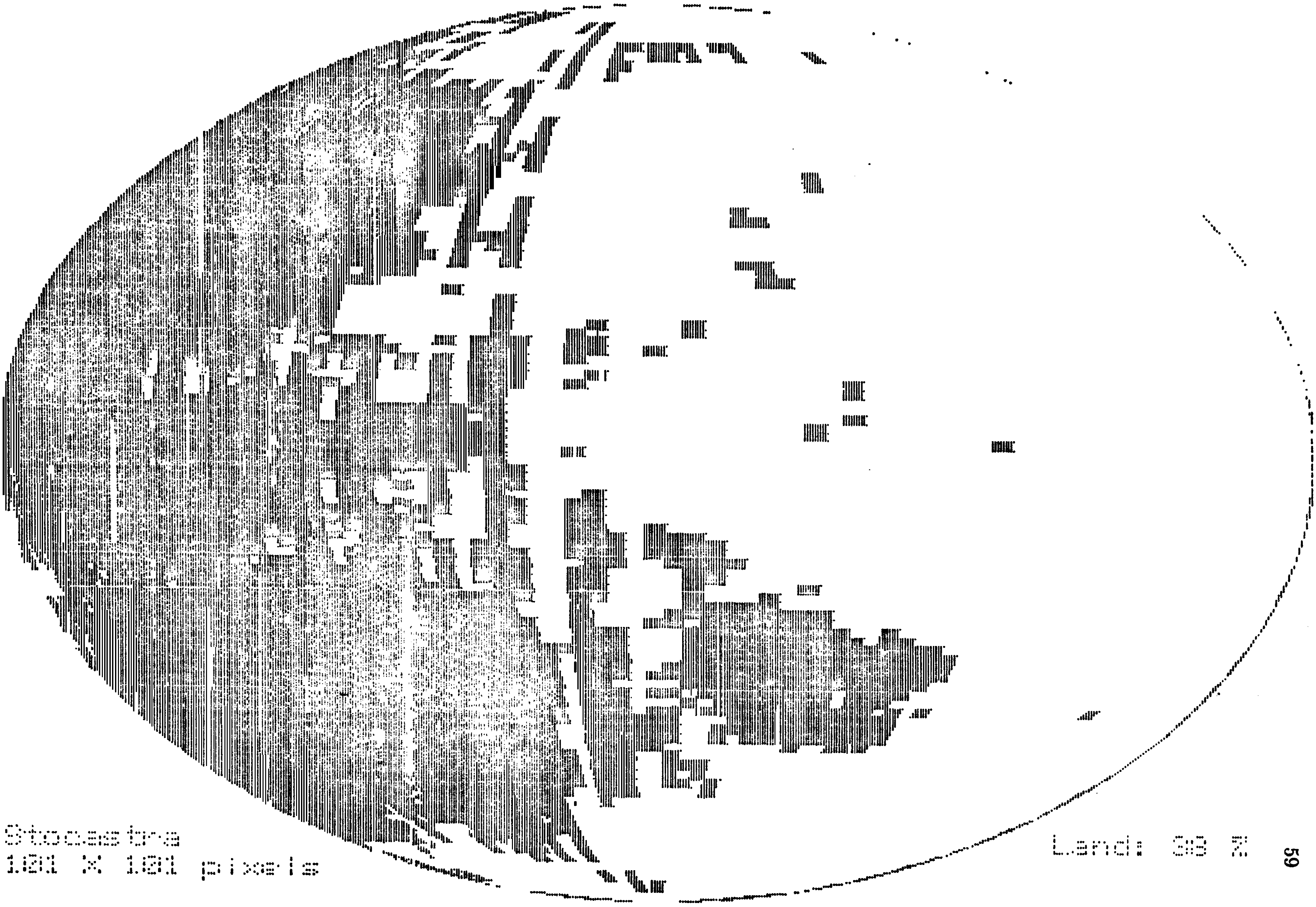
Using this, they developed an algorithm for generating approximate Brownian motion on a plane, with an efficient method that converges to true Brownian motion as in the problem statement. They mapped this onto a unit disk and then a sphere. Their efforts were rewarded when a bystander, unaware of what was being displayed, looked at their screen and said, "That's the Caspian Sea."

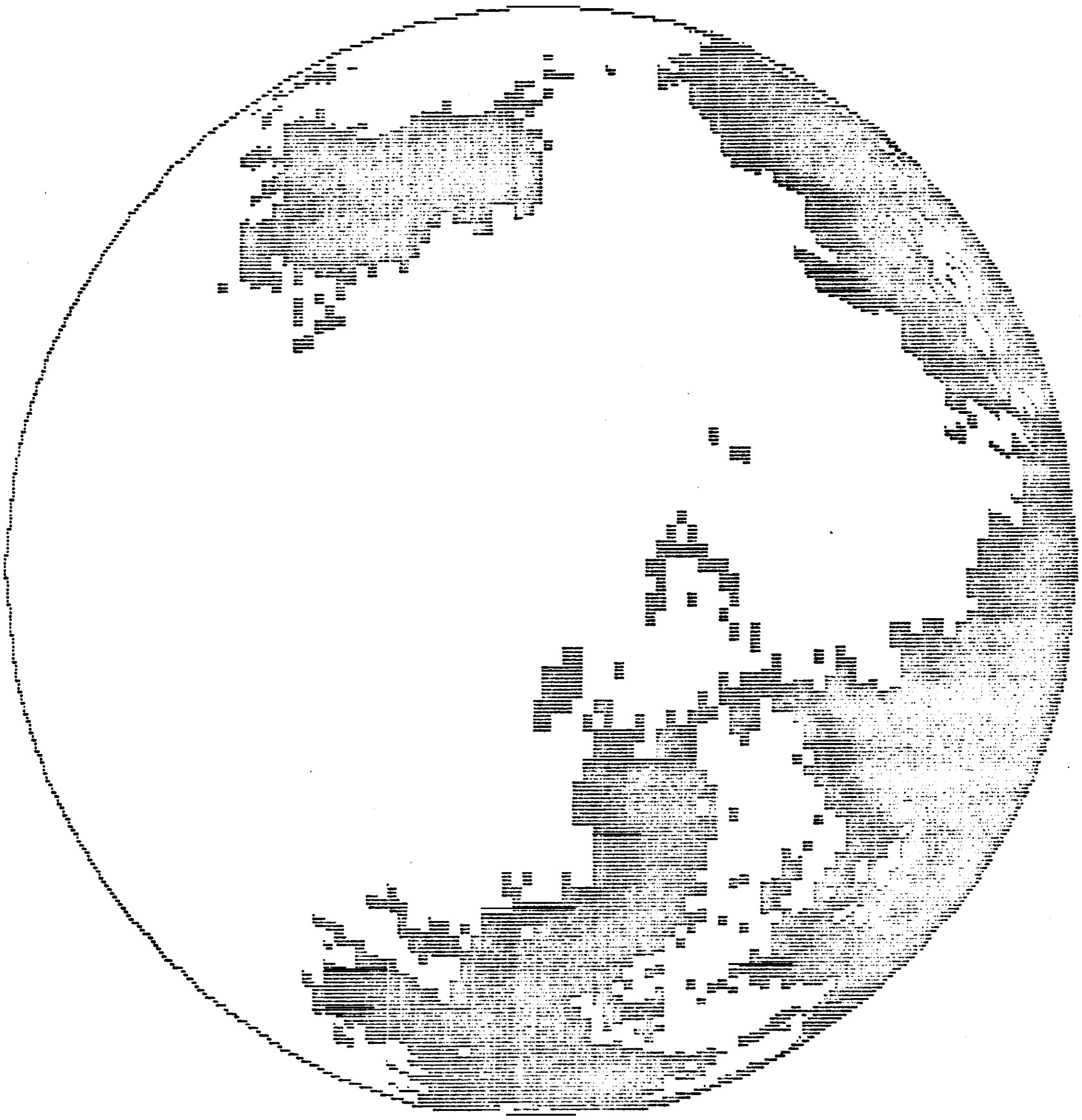
The next six pages of this report contain maps of Stochastra produced by the programs, in the following order: DMC, JCM/AIS, RJM, JEH/SSK/DEF, JF/DPH/TPM and LAH/ARK/YOM.

This is Dave Chalberg's Stochastra!

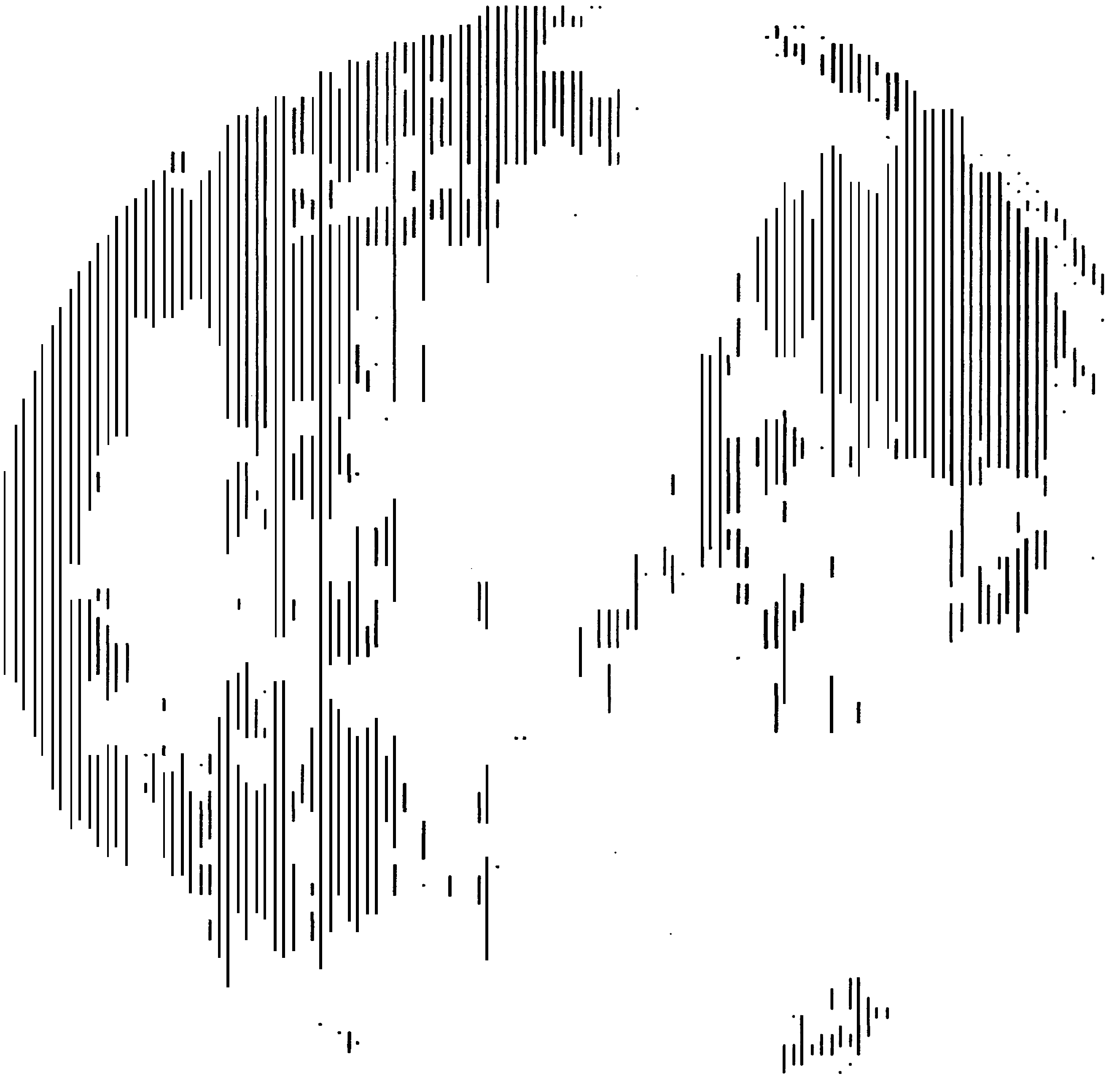
Stochastra m = 128, n = 128 rotation = 90, threshold = -0.001001

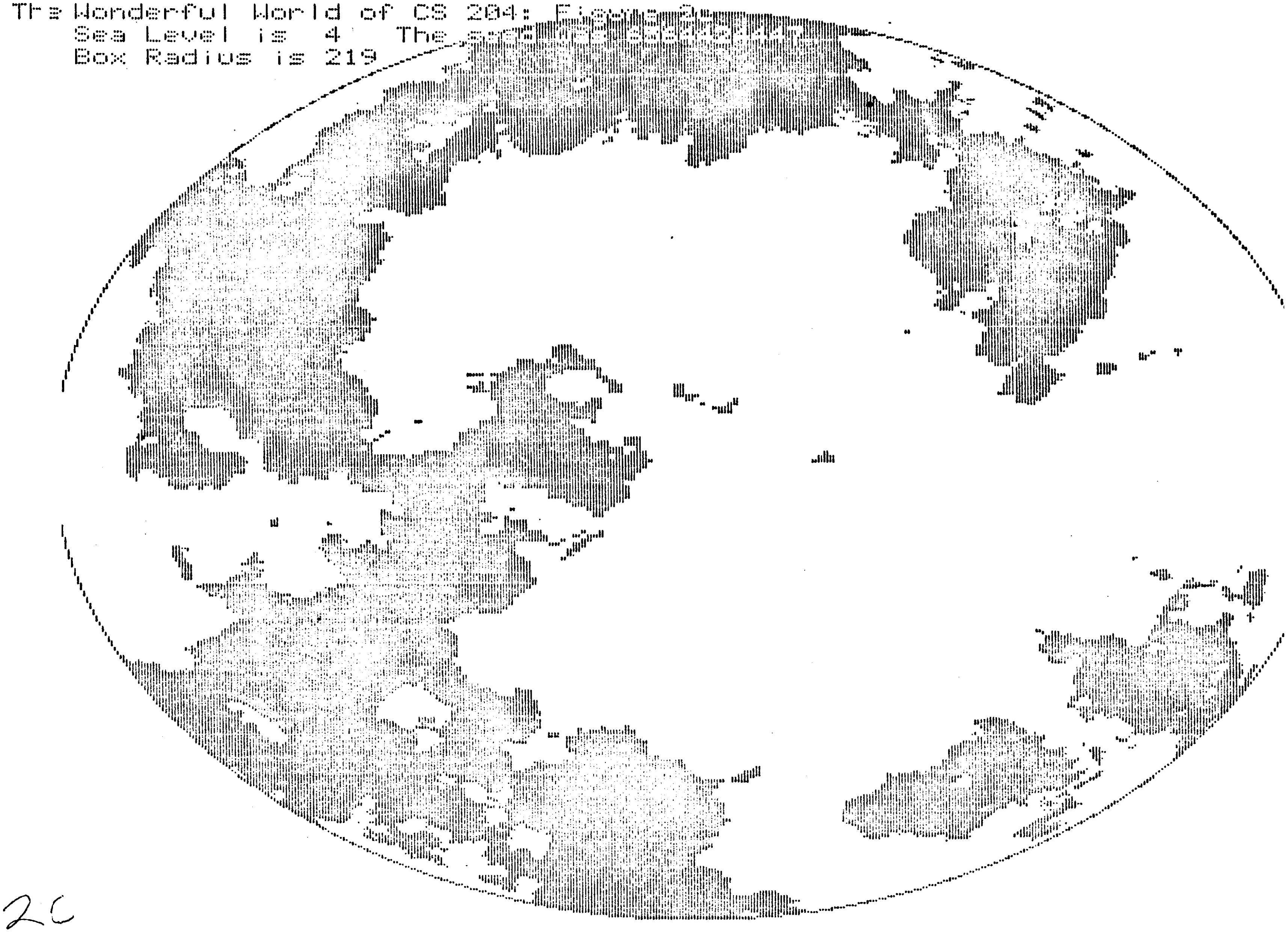



Stocastra
101 X 101 pixels
Land: 38 %



Seed = 1234
View = .25




The Wonderful World of CS 204: Figure 2
Sea Level is 4   The [illegible]
Box Radius is 219

2C

Problem 5: Question Answering

The goal of this problem is to obtain a computer program that answers questions put to it by naïve users, deriving the answers from a small database that has information about the occupants of Jacks Hall. For example, your program should be able to answer questions like the following:

> Who is in room 328?
> What is Ullman's home phone number?
> Where is Professor Golub's office?

And possibly also questions like this:

> In what room is extension 70481?
> Do you know anybody named Ursula?
> Who are the secretaries on the third floor?
> Verstehen Sie Deutsch?

Of course, we have to draw the line somewhere, since no program can be expected to answer all conceivable questions satisfactorily.

> What was the stupidest question you were ever asked?

The trick is to find a way to give sensible answers to as many questions as possible, without an overly complicated program.

Assume that the user of your program has had little or no previous experience with computers, and is not interested in learning a bunch of rules. The user just wants answers to questions, and knows only how to type a question at the terminal.

December 8 will be Demonstration Day, on which your program should be up and running. The entire class period on that day will be devoted to trying out each other's solutions to this problem.

A database that holds all the relevant information about Jacks Hall people will be available to everybody in the class. Its precise format will be clarified later, but basically it will consist of a number of records containing a name and optional further attributes: room number, office phone, home phone, title, nickname, home address, birthday.

Class notes for Tuesday, December 1

DEK introduced the final problem today by explaining that he intended it partly as an exercise in "human engineering": designing a program that a naïve person can use without having to read a manual or learn a specialized language. The data for the problem, which was presented as a file available on SAIL and SCORE, consisted of about 240 lines, representing the occupants of Margaret Jacks Hall. DEK said not to worry about whether this data was completely accurate; we can just assume that it is correct for the purposes of this problem, even if there are changes since the file was compiled. DMC said that a more general program could update the database from the dialog with its users, but DEK suggested that this would add unnecessary complication to the problem, and was too much to expect given the short time available.

As a general approach, DEF related an experience that he had with a similar program, which ignored most of the sentence structure of its input and just looked for keywords. DEK called the ignored text "noise words," and said that they included many common words such as 'is', `'the'`, etc. JLM pointed out that one class of words that could not be placed in this category are words that express negations. When such a word is encountered, some analysis of the input is needed to determine what the negation refers to. DPH thought that it might still be reasonable to ignore this, at the cost of sometimes printing out more information than the user requested, but JF said that this might cause the program to print out the entire database. RJM suggested the idea of "links" between keywords and the objects that they represent, i.e., between nouns and things, and between verbs and actions.

DEK compared the dialog between such a program and its user to the experience of trying to communicate in a foreign language that one does not really understand, and said that in such a situation, between people there always seems to be a way to get the meaning across. He said that we should not assume that the user of our program is an antagonist, trying to cause it to fail, since no matter where we draw the line between what the program can and cannot do, there will be ways to confuse it.

KPB said that the vast majority of questions asked would consist of specifying some of the fields in a record and asking for others. Therefore, he suggested that we concentrate on discovering what the most common English phrases to express this situation are. DEK decided to look at the example questions in the problem description.

```
Who is in room 328?
```

In this question `'who' is a` keyword, which indicates a request for the name of a person (or persons). 'Room' is also a keyword, and has the property that if it is followed by a number, then that number is the given value of the room field in the record(s) requested. Some people disputed the interpretation of `'who',` saying that it might have other meanings. For example, if the question was

```
Who is Don Knuth?
```

the desired answer would be `'Professor'.` In response to some who thought that the word `'who'` could be ignored, RJM said that it was necessary, since a question such as

```
What phone is in room 402?
```

requests information other than the name field. In fact, the database being used has a separate line for each of the occupants of 402, giving the same phone number. Thus we would want to be sure that only one instance of the information was printed out. DEK mentioned that when a question is too complicated for the program to handle, it should 'fail gracefully' with a polite response such as 'I don't know', rather than printing an obscure error message.

The next question,

`What is Ullman's home phone number?`

introduced some more interesting problems. In this case, the word 'home' is important, since the database contains both the office phone number and the home phone number for each person. A problem arises when the user does not specify which phone number he wants. There are two ways to handle this: either the program can give both phone numbers, or it can ask the user which one he wants to know. The first solution is easier, of course, and most of the class decided to do this, though it was pointed out that you need to be careful not to print out too much information by generalizing this example. JF pointed out that in this question, the word 'what' is important in that it requests information, but it does not tell what information is being sought, as is the case with 'who' or 'where'.

Now DEK asked how the word '**Ullman**' in the question would be recognized as important by the program. First of all, it had to be separated from the apostrophe and 's' at the end. More generally, the program might have to deal with plurals, verb. endings, misspellings, etc. Almost everyone was against the idea of dealing with spelling correction in their programs; if a spelling error was made, the program could give an 'I don't know' response. We could consider the names of people and other objects in the database to be keywords, and DMC suggested using keywords as a "sieve" to eliminate lines in the database that do not apply to the question being answered. JLM disagreed with the idea of making keywords out of words such as names, since if we later want to add a spelling checker, we don't want '**Ullman**' fixed in the program as the only way to recognize this name.

The next issue raised by DEK was the question of access to the database. He contrasted a simple approach of linearly scanning the entire file for each transaction to using a more complex approach that might have average- or worst-case access time of $O(\log n)$, where n is the number of database entries, and asked the class which they thought would be appropriate. AH noted that binary search would be an efficient way to look up a name, since the database was already sorted in order of last names. RJM said that some sort of hashing would allow efficient access given any of the fields. KPB said that this issue could divide the problem into those cases where all the data can fit in main memory, and those where access to secondary storage is needed. TPM stated that questions of database access were not the important part of this problem at all, and he preferred to concentrate on the interface with the user. DEK agreed with this sentiment, saying that even though his own research often deals with efficient algorithms for data accessing, a database with only about 200 lines is far too small to consider using such techniques. He said that it would be quite acceptable to read in the entire database once at the beginning of the program, and to search it linearly for each request.

Returning to the discussion of analyzing the input, DEK said that the "sieve" approach could be generalized to make every word a sieve of some sort. Noise words, such as 'is', would allow anything in the database to pass through, and names such as 'Ullman' would allow only one record. JF added that some of the sieves would operate on fields instead of records; for example, the word 'phone' would choose two of the fields and eliminate the others. DMC said that we would have to be careful with this when the question contains the word 'or', which could ask whether either of several sieves apply. JEH provided the example

What people are in rooms 326 or 328?

which illustrates this point.

RJM asked how to deal with a request for information from more than one field in a record. JF pointed out that in the model discussed earlier, this would correspond to having more than one field marked as unknown, along with the values of the fields that were specified. KPB asked whether we should include information picked out by the word 'not', such as

Who does not have a telephone?

DEK said that in most cases, 'not' is used in a construction of the form '. . . and not . . .', which places a limit on the total amount of output. Even if a large amount of output **is** generated for a particular question, RJM suggested asking the user if he really wants to see it whenever it exceeds a threshhold.

The class spent a while discussing the ways in which the word 'and' presents difficulties. To most mathematicians, 'and' indicates the conjunction of two conditions, while in everyday English, it is more often a disjunction. Thus the phrase 'men and women' refers to all people, not the empty set!

Several more comments were made as the class ended. JEH said that regarding spelling errors, if the program encountered a situation where it could not find any records that matched keywords found in the input, then it could report back to the user with the suggestion that a word was probably misspelled. TPM raised the question of strange input containing unrecognized words, such as 'Pray tell me . . . '. In such a case, the program would not know that 'pray' is a noise word. JCM suggested that if the program can provide an answer by ignoring words that it does not recognize, then it should do so; otherwise, it could go check for spelling errors if it has this ability. AH pointed out that a useful type of spelling "error" detection is the use of common nicknames instead of full first names.

Class notes for Thursday, December 3

DEK began by writing on the blackboard the question

Are there no people on the first floor?

This question can be answered 'Yes' or 'No', but whichever answer is given the user wouldn't know for sure what the computer thought it was saying. This is one that most programs would not be able to handle correctly, because of its ambiguity. Therefore, an answer of 'Yes' or 'No' could be misleading, since it carries the implicit conclusion that the program was able to understand the question. Therefore, DEK said he preferred to have all answers be statements of fact such as

```
There are no people on the first floor.
```

Answers like 'Yes' and 'No' are based on the context in which they appear, and DEK said that dealing with context (including proper understanding of prepositions) is an important aspect of artifical intelligence programs that carry on dialogs. He considered such refinements to be beyond the scope of Problem 5, but he wanted to mention their importance in real-life applications.

Another question that might cause problems is

```
Is Knuth in room 326?
```

Since the database contains the information that Knuth's office number is 328, several possible answers would be

```
I don't know.
Knuth is in room 328.
Phyllis Winkler is in room 326.
Nothing in my database matches both Knuth and room 326.
```

JLM mentioned that in a seminar he had attended several months ago, the idea of a 'coöperative' database management program was discussed. This sort of program checks for implicit assumptions in the questions it is asked, and if any of these are false, they are reported back to the user as a probable error. Thus, given the question

```
Is Knuth in room 726?
```

a coöperative program would answer that there is no seventh floor, rather than simply 'No', since the question contains an incorrect assumption.

DMC asked whether there was any way to decide if a question called for a yes/no response. JCM suggested that if the question gives values for some fields, but does not name any fields without giving values, then a yes/no answer is appropriate. DEK said that even in this case, he preferred that the program output a statement, such as the ones given above. KPB disagreed with this, saying that one thing that makes computer programs seem "dumb" is that they are too verbose when the user expects a simple answer. He said that to recognize yes/no questions, the program could check whether the first word of the question is 'Is', 'Are', 'Does', 'Do', 'Can', etc.

At this point, DEK asked whether anyone wanted to share with the class the strategies that their programs were using. KPB outlined his, which involved three main phases in the program. First, it reads the database and creates a list structure (the program is written in LISP) to represent the information. This involves some simple deduction, to separate the lines representing people from those naming machines in rooms, or rooms with no named occupants. Also, the floor of each room is determined from the room number, so that questions can include information about floors of the building.

The second phase of KPB's program reads the user's input. Based on the first word of the question, it categorizes the question as being one of (1) a request for information, (2) a question requiring a yes/no response, (3) a "how many" question, or (4) a "describe" question, asking for all fields in one or more records. The yes/no questions are further divided into two types: (a) a "does he have" question, and (b) an "is it true that" question. Category (1), which uses information in the question to fill some fiklds, and requests some of the other fields, is the default when a question does not fall into any of the other categories.

The third phase computes the answer, and produces an English sentence for it. If there are no keywords recognized in the entire question, the program will say `'I don't know'`. Because it does not analyze its input for syntactic correctness, KPB said that it will actually provide meaningful responses to input such as

```
Don Knuth what room what phone?
```

YOM described his program's basic structure. First, it reads the database and does some preliminary processing (such as converting all letters to lower-case, for easier pattern matching). Then, when a question is typed in, the first step is to determine what is being asked. The second stage is to find what knowledge is present in the question. The third phase produces an appropriate response. YOM asked whether anyone knew how efficient the set-manipulation features of SAIL are. DEK said that he had never used them himself, but thought that they should be fast enough for this problem in any case. RJM had done some of his program using these features initially (though he later switched to LISP), and said that there was no noticeable slowness at all.

RJM and JLM, who were working together, now described their program. Given the question 'Are there no people on the first floor?', it would eliminate some words as "noise," and end up by producing the list

```
(EQUAL 0 ON PEOPLE FLOOR FIRST).
```

This sentence, however, was not chosen as an example of normal input, so two better ones were given:

```
Whose phone is 7-4132?
Who was born in June?
```

The first one becomes the list `(WHO PHONE IS (PHONE 7 DASH 4132))`; the inner list at the end is a result of the scanner's recognizing the structure of the input representing a telephone number. The second pass of the program recognizes that `'PHONE'` is both a word in the question and a field that has been supplied, so it converts the list to

```
(WHO (PHONE 7 DASH 4132)).
```

This causes the program to return the name of a person, because the list begins with 'WHO'.

The second question becomes (WHO (BIRTHDAY (DATE JUNE))), in an an analogous manner. JLM explained that the program could handle more complicated queries, such as

Whose birthday is before June?

which would be converted to (WHO (BIRTHDAY (DATE: RANGE NOW (DATE JUNE)))). The program does these conversions by means of a grammar. For example, the production that does the conversion just mentioned is

BEFORE (<DATE>) → (DATE:RANGE Now 2).

Here 2' means the second element at the left of the arrow. You start with the user's given sentence and see if any productions can be applied starting at the first word; if you can, you replace one or more words that match the left-hand side of the production by the quantity or quantities on the right, and start over. If not, you try matches that start at the second word, and so on, until you find that no more productions can be applied.

ARS asked about questions that may have conflicting information. For example, in the question

Who was born before Knuth?

the word 'before', which normally introduces a date, is followed by a name. He said that part of the program could resolve this conflict by trying to produce the desired information (a date) from what is present. In this case, the record with the name 'Knuth' has a date (his birthday), and so this could be used as the object of 'before'.

Returning to the discussion of grammars, DEK said that a production-driven approach like this is useful because it is a fairly straightforward programming task to write a parser. The hard part, though, is deciding what the productions should be. Once the grammar gets large, it is very hard to determine how the productions interact, and unexpected results may arise. Also, if you test the grammar by choosing examples and making sure they are accepted, then it is easy to "cheat" by introducing productions just for the sake of fixing bugs in a few examples that you want to demonstrate.

DEK mentioned that an extension of the basic grammar approach is one that may have several productions that are applicable at any time, with "weights" assigned to each so that of several possible parses for a sentence, we can decide which one is most likely to be "correct." DEK also noted that you can add productions as you gain more experience with your program. It turns out that the the power of production systems allows one to handle special cases in the language by "programming" tricks; he recommended against doing this, since it makes the grammar hard to understand and modify. Each production chosen should have a natural interpretation and be meaningful in the context of the rest of the grammar.

JLM pointed out that the use of idioms in English can be captured in a production system; he felt that this was a legitimate use of productions, but that those productions should be made separate from the others in the grammar.

RJM told about one more feature of his program-a “hierarchy” of associations. Each keyword is given not only an action, but also a corresponding priority. If two keywords have the same priority, the program takes the union of their results; this is intersected with the result of keywords of higher priority. For example, ‘who’ has a higher priority than ‘phone’, so the phone number that is output by the action of ‘phone’ is not included in the final output. Another example of this is that the program does not print out a full name unless it is required by the question.

JLM mentioned another advantage of a production grammar: partial results can be used even when the sentence as a whole cannot be parsed, since production parsing operates “bottom up.” By contrast, a “top down” method gets nowhere unless it can parse the whole sentence.

Class notes for Tuesday, December 8

Today was Demonstration Day, and seven groups had programs running **and** ready to show to the class. DEK (with help from his wife) had prepared a list of questions that he wanted each program to try out. These included several that he did not expect most programs to interpret correctly. In addition, each group provided one question to “show off” their program, and see if the others could handle it.

(A note to the reader: some of the transcripts below have been edited to save space. All of the answers are from the programs, though.)

The first program was by KPB and AH. It was written in MACLISP.

```
>Who is in room 306?
  James Wilkinson

>What is Ullman's phone number?
  office phone- 7-1612    home phone- 494-2667

>Where is Gene Golub's office?
  216

>What is his home address?
  I don't know. Ask a human.
```

This question was designed to fail, since connecting the word “his” with the previous question is more complicated than anybody was supposed to attempt.

```
>What is Gene's home address?
  328-7064
```

Since address information is not part of the database, the word “address” is ignored, but “home” then refers to the home phone number.

```
Where is the ladies' room?
  I don't know. Ask a human

>Can I use a computer here?
```

```
  I don't know any.
```

AH explained this strange answer by saying that the "I" matched the database entry for "Andrew I. Shore." When the question was changed to

```
>Can i use a computer here?
  Almanor
  Diablo
  Diego
  Inyo
  Narin
  Nojave
  Mono
  Nonterey
  Palomar
  SAIL
  SCORE
  Toro
  Yolo
  Yuba
```

it provided a list of all the computers in the building.

```
>Where are the terminal rooms?
  320
  460
  408
  343
  335

>Is there a place to buy food, drink?
  I don't know. Ask a human.

>Where is the departmnt secretary's office?
  Carolyn Tajnai - 210
  Marilynn Walker - 210

>Is there a xerox machine that i can use?
  yes
```

The "I" was put in. lower case to make the previous question work.

```
>Where is the Xerox on the fourth floor?
  433

>What is Nancy's last name?
  CSD Receptionist

>How many students are at extension 7-0926?
  8

>Is today anyone's birthday?
  I don't know. Ask a human.

>What are Dave's birthday and phone number?
  Dave Arnold - office phone- 7-3796 home phone- 962-0376
```

```
birthday- December 16
  David Brown - office phone- 7-2462 home phone- I don't know
birthday- June 12
     ...
```

This continued, printing out information for anyone named "Dave" or "David."

```
>Who shares an office with Dave Foulser?
  316

>Is Donald in room 328?
  yes

>What secretaries were born in Novem.?
  I don't know any.
```

The strange abbreviation was not anticipated. With a slight modification, it will work.

```
>What secretaries were born in Nov.?
  Jayne Pickering
  Marilynn Walker

>Where are the Altos?
  I don't know any.
```

The program can't handle both plurals and capitalization in the same word.

```
>Where are the altos?
  Almanor - 241
  Diego - 225
  Inyo - 433
     ...

>Qui est sur la deuxieme etage?
  I don't know. Ask a human.

>Which students in Lia's office do not have home phones?
  room 416A home phone- 493-4952
```

The next program was by JCM, ARK, and AIS. It is written in C.

```
> who is in room 306?
The name of people/places/things in room number 306 is:
James Wilkinson ,

>
I matched every record in the database --
You probably want to rephrase your question

> What is Ullman's phone number
The office phone and home phone of Ullman is:
7-1512 494-2667

> Where is Gene Golub's Office
The room of Gene Golub is:
216
```

```
> What is his home address?
I matched every record in the database --
You probably want to rephrase your question
I didn't understand the word(s):
 ‘his’, ‘address'

> What is Gene's home address?
The home phone of Gene is:
328-7064

> Where is the ladies' room?
I matched every record in the database --
You probably want to rephrase your question
I didn't understand the word(s):
 ‘ladies''

> Can I use a computer here?
Yes

> Where are the terminal rooms?
The rooms of the Terminal Rooms are:
225

343

408
460

320

> Is there a place to buy food, drink?
Yes
```

That's interesting! Let's see if it can tell us where such a place is.

```
> Where
I matched every record in the database --
You probably want to rephrase your question

> Where is the department secretary's office?
The rooms of the Secretarys are:
326
210
210
030
206
214
436
340
304
358

> Is there a xerox machine here that I can use?
Yes

> Where is the Xerox on te fourth floor
```

```
The room of the Xerox Graphics Printer on the Fourth floor is:
433


> What is Nancy's last name?
The last name of Nancy A. is :
Dorio ,

> How many students are at extension 7-0926?
No
I didn't understand the word(s):
'many', 'extension'

> is today anybody's birthday?
I matched every record in the database --
You probably want to rephrase your question
I didn't understand the word(s):
'today', 'anybody's'
```

A more specific question was given.

```
> who was born on december 8?
The name of people/places/things with birthday December and 8 is:
Prancing Pony ,

> What are Dave's birthday and phone number?
The office phone, home phone, birth day and birth month of Andrew I. is:
7-0926, 328-7965, 30, September
```

The reason for that response is the word 'and', which matches 'Andrew' in the database. If we remove 'and' from the question, we get the following response.

```
> What are Dave's birthday & phone number?
I do not understand your question --
There is no such information in the database

> Who shares an office with Dave Foulser?
I do not understand your question --
There is no such information in the database
I didn't understand the word(s):
'shares', 'an', 'with'

> Is Donald in room 328?
Yes

> What secretaries were born in Novem.?
The birth days and birth months of the Secretarys are
birthday not known
4 , November
21, July
13, April
23, September
birthday not known
26, November
14, December
17, September
```

```
birthday not known
```

This program is also unaware of the strange way of abbreviating 'November'.

```
> What secretaries were born in nov?
The birth days of the Secretarys with birthday November are:
4
26

> Where are the Altos?
The rooms of the Altos are:
460

460

460

030C
330

450

460

408

450

433

225

241


> Qui est sur la deuxieme etage?
Permettez-moi a repondre en Anglaie:
The names of people/places/things on the Second floor are:
Dawn Yolton ,
name not known ,
Terry Winograd ,
Bill White ,
Marilynn Walker ,
Allan Terry ,
name not known ,
Carolyn Tajnai ,
Mark Stefik ,
first name not known Spruce ,
David E. Smith ,
Susan Seabrook ,
Betty Scott ,
Irmgild Schack ,
Barbara Roberts ,
Glenn Rennele ,
Mark Peairs ,
Thien-Huong Palmer ,
Gordon Novak ,
Penny Nii ,
Jock Mackinlay ,
Doug Lenat ,
Barbara R. Laddaga ,
John Kunz ,
Ingeborg M. Kuhn ,
Dianne Kanerva ,
Susan Hill ,
Milt Grinberg ,
Russell Greiner ,
Lynn Gotelli ,
Jose L. Gonzalez ,
Gene Golub ,
Phil Gerring ,
Mike Genesereth ,
Peter Friedland ,
Jeff Finger ,
Edward Feigenbaum ,
David A. Eppstein ,
Bob Engelmore ,
Nancy A. Dorio ,
first name not known Diego ,
Linda Gail Demichiel ,
Claudia Dedo ,
name not known ,
Paul Cohen ,
William Clancey ,
Bruce Buchanan ,
Jake Brown ,
Harold Brown ,
Denny Brown ,
James Bennett ,
Avron Barr ,
Rene Bach ,
first name not known Almanor ,
Nelleke Aiello ,

> Which students in Lia's office do not have a home phone?
The rooms and home phones of Lias the Students are:
311 366-9004
416A 493-4952

7 who is in 311?
The names of people/places/things in room number 311 are:
Stephen Nash ,
William Gropp ,

> what time is it?
Tue Dec 8 11:43:59 PST 1981

> what is your favorite color?
red

> what is the airspeed of an unladen swallow?
African or European?
```

```
> Goodbye
```

The third group to demonstrate their program was DMC, SSK, and DPH. Theirs was written in SAIL.

```
Your question? who is in room 308

The following are in room 306:
james Wilkinson
Your question? what is ullman's phone number

jeff ullman's office phone number is 7-1512.
Your question? where is gene golub's office

I do not know.Nothing like that in my data
```

The word "off ice" is not known to the program. Maybe it will work if we try "room."

```
Your question? where is gene golub's room

I do not know.Nothing like that in my data
Your question? what is his home address

I am confused.Please rephrase your question
Your question? what uis gene's home address

gene golub's home phone number is 328-7064.
Your question? where is gene

gene golub's room is 216 Margaret Jacks Hall.
Your question? where is the ladies' room

I am confused.Please rephrase your question
Your question? can i use a computer herre

I am confused.Please rephrase your question
Your question? where are the terminal rooms

I am confused.Please rephrase your question
Your question? is there a place to buy food,drink

I am confused.Please rephrase your question
```

A rephrasing of one of the previous questions did produce a response.

```
Your question? where are the terminals

The following rooms contain terminal room :
225
343
Your question? where is the department secretary's office

I am confused.Please rephrase your question
Your question? where is the department secretary

The following rooms contain department secretary :
```

```
2 10
2 10
Your question? is there a xerox machine here that i can use

I am confused. Please rephrase your question
Your question? where is the xerox on the 4th floor

I am confused.Please rephrase your question
Your question? who is the xerox on the 4th floor

The following are on floor 4:
 xgp
Your question? what is nancy's last name

I am confused.Please rephrase your question
Your question? who is nancy

nancy dorio is a csd receptionist .
Your question? how many students are at extension 7-0926

The following people have phone number 7-0926:
andrew shore
ralph saaveddra
yoram moses
timothy mann
sanjay kasturia
david helmbold
david chelberg
richard anderson
Your question? what are dave's birthday and phone number

I am confused.Please rephrase your question
Your question? is donald in room 328

yes,donald knuth's room is 328 Margaret Jacks Hall.
Your question? what secretary born in nov

The following people were born in nov:
jayne Pickering
```

The program did not print "marilynn walker" because her title is given in the database as "department secretary."

```
Your question? where are the altos

The following rooms contain alto :
4 60
4 60
4 60
0 30
3 20
4 50
4 60
4 08
4 50
4 33
```

```
2 25
2 41
Your question? which students in lia's office do not have a home phone

I am confuaed.Please rephrase your question
```

The next program, also in SAIL, was by JF, LAH, and YOM.

```
@ithink

Welcome to the wonderful world of CS204, foolish human.

You may type questions about Margaret Jacks Hall.
I may answer them.

When you have had quite enough, type "abort".

Ask me a question, if you dare!
 ?>who is in room 306?
James Wilkinson is in room 306.

Ask me a question, if you dare!
 ?>
Excuuuuse me
```

(The program prompts each time with "Ask me a question, if you dare!;" this has been removed from here on.)

```
?>what is ullman's phone number?
Jeff Ullman has office phone number 7-1512 and has home
        phone number 494-2667.

 ?>where is Gene golub's off ice?
Gene Golub is in office 216 and is on the second floor .

 ?>what is his home address?
This has nothing to do with the database.

 ?>what is Gene's home address?
I don't know any such person.

 ?>who is Gene?
I don't know any such person.

 ?>where is the ladies' room?
Barbara R. Laddaga  is in office 248 and is on the second
        floor .
```

Apparently, the string "lad" caused that response.

```
?>Can i use a computer here?
Ralph Gorin  was born on december 24 and is a director of
        computer and is in office  422 and has office phone number
        7-3236 and has home phone number 326-5879 and is on the fourth
        floor .
Chris Tucci was born on december 10 and is a sail user and
        is in office  020e and has office phone number 7-2896 and has
        home phone number 327-6071 and is in the basement .

 ?>where are the terminal rooms?
Invalid index for array ANSWERS. Index no. I, value is 241
Called from 416742 Last SAIL call at 416734
?IReply [CR] to continue,
[LF] to continue automatically,
"D" for DDT, "E" to edit, "B" for BAIL,
"X" to exit, "S" to restart
?s
```

At this point, the program had to be restarted.

```
?>is there a place to buy food,drink?
This has nothing to do with the database.

 ?>where is the department secretary's office?
Invalid index for array ANSWERS. Index no. I, value is 241
Called from 416742 Last SAIL call at 416734
     ...
```

Another question caused the same error.

```
?>is there a xerox machine here that i can use?
Yea:

 ?>where is the Xerox on the fourth floor?
Invalid index for array ANSWERS. Index no. I, value is 241
Called from 416742 Last SAIL call at 416734
     ...

 ?>What is Nancy's last name?
Nancy A. Dorio .

 ?>How many students are at extension 7-0926?
There are 8.

 ?>What are dave's birthday and phone number?
Dave Arnold was born on december 16 and has office phone
        number 7-3796 and has home phone number 962-0376.
Dave Marimont waa born on   and has office phone number
        7-3796 and has home phone number 864-6989.

 ?>is donald in room 308?
No,
```

A mistype, but it shows that the program works!

```
?>is donald in room 328?
Yes, Donald E. Knuth is in room 328.

 ?>what secretarie were born in Novem?
Peter Blicher was born in the month of november.
Denny Brown was born in the month of november.
Jim DaviDson was born in the month of november.
Kevin Karplus was born in the month of november.
Gabriel M. Kuper was born in the month of november.
Amy Lansky was born in the month of november.
Clayton wai Mock was born in the month of november.
Jayne Pickering'was born in the month of november.
David Todd was born in the month of november.
Marilynn Walker was born in the month of november.

 ?>what secretaries were born in Novem?
Jayne Pickering is a secretary and was born in the month of
        november.
Marilynn Walker is a department secretary and was born in the
        month of november.

 ?>where are the altos?
Invalid index for array ANSWERS. Index no. I, value is 241
Called from 416742 Last SAIL call at 416734
     ...

 ?>qui est sur la deuxieme etage?
This has nothing to do with the databaae.

 ?>which students in lia's office do not have a home phone?
I don't know any such person.

 ?>abort

I'm meeeeeelting . . . . . . . .
  [ Bus Error - Core Melted ]
Ding, dong . . . . .
End of SAIL execution
```

Next came TPM and RJA, who had also used the SAIL language.

```
Welcome to the RA/TM question answering system.
Do you want me to read questions from a file? n
Ok. You can type a question whenever you see the '#' sign.

# who is in room 306?
James Wilkinson is in 306.

# what is Ullman's phone number?
Ullman has office phone number 7-1512.

# Where is Gene Golub's office?
Gene Golub is in 216.

# What is his home address?
I don't know what you mean.
```

```
# What is Gene's home address?
Gene is a Professor.

# Where is the ladies' room?
I don't understand that question.

# Can I use a computer here?
Almanor.
Diablo.
Diego.
Inyo.
Marin.
Mojave.
Mono.
Monterey.
Palomar.
SAIL.
SCORE.
Toro.
Yolo.
Yuba.

# Where are the terminal rooms?
Room 343.
Room 226.

# Is there a place to buy food, drink?
Yes.

# Where is the department secretary's office?
Room 210.

# Is there a xerox machine here that I can use?
Yes.

# Where is the Xerox on the fourth floor?
Room 433.

# What is Nancy's last name?
Nancy.

# How many students are at extension 7-0926?
There are 8.

# Is today anybody's birthday.
Yes.
```

In view of the previous "Yes" answer, the class wondered whether this was just an automatic response. So a more specific qustion was typed in.

```
# Who was born December 8?
Prancing Pony was born on Dec 8.

# What are Dave's birthday and phone number?
Dave Arnold has office phone number 7-3796 and was born on Dec 16.
David Brown has office phone number 7-2462 and was born on Jun 12.
David Chelberg has office phone number 7-0926 and was born on Feb 16.
David Eppstein has office phone number 7-3324 and was born on May 27.
David Foulaer was born on Oct 4.
David Fuchs has office phone number 7-1646 and was born on Sep 12.
David Helmbold has office phone number 7-0926.
David Lowe has office phone number 7-3796.
Dave Marimont has office phone number 7-3796.
David Smith has office phone number 7-1809.
David Todd has office phone number 7-0445 and was born on Nov 19.
```

Note that no mention is made of information not present in the database. The following question demonstrates that "Who" returns information about a person, while "What" is ignored, since TPM and RJA found it to have too many possible interpretations in different situations.

```
# What is David Eppstein?
I don't have that information.

# Who shares an office with Dave Fouleer?
David Foulser is in 316.

# Is Donald in room 328?
Yes.

# What secretaries were born in November?
Nov 26.
Nov 4.

# Who are the secretaries born in November?
Jayne Pickering was born on Nov 26.
Marilynn Walker was born on Nov 4.

# Where are the Altos?
Room 241.
Room 226.
Room 433.
Room 460.
Room 408.
Room 460.
Room 320.

# where are the alto wizards?
Room 030C.

# Qui est sur la deuxieme etage?
I don't know what you mean.

# Which students in Lia's office do not have a home phone?
I cannot answer that question.
```

TPM decided to type in a few more questions.

```
# why is the Dover down?
I cannot answer that question.
```

```
# who is the basement
Dave Arnold.
Harlyn Baker.
Tom Binford.
Peter Blicher.
David Brown.
Don Coates.
Mark Crispin.
Martin Frost.
Ron Goldman.
Roy Hayes.
Peter Hochschild.
Lassen.
Sidney Liebes.
David Lowe.
Michael Lowry.
Dave Marimont.
Allan Miller.
Clayton Mock.
Mark Roberts.
SAIL.
George Schnurle.
SCORE.
Marianne Siroker.
Chris Tucci.
Rick Vistnea.

# who are the research people?
I don't know what you mean.

# who are the r & d engineers?
David Brown.
Roy Hayes.
George Schnurle.

# who is visitor, ullman?
There are none that I know of.

# who is a visitor?
Peter Blicher.
James Bunch.
Elise De Doncker.
Mike Harrison.
Ruth Levinson.
Fumichiro Maruyama.
Ernst Hayr.
E.O. Milton.
Gordon Novak.
Mihoko Okada.
Vaughan Pratt.
David Todd.

# help
I know all sorts of things about the people, rooms, and
    computers in Margaret Jacks Hall.  Ask me questions in
    simple English, and I will try to answer.  I am good with
    birthdays, phone numbers, room numbers, and the like.
    In fact, I'm smarter than my programmere!

# help
I  know all sorts of things about the people, rooms, and
   computers in Margaret Jacks Hall.  Ask me questions in
   simple English, and I will try to answer.  I am good with
   birthdays, phone numbers, room numbers, and the like.
   I'm not so good at remembering faces.

# help
I know all sorts of things about the people, rooms, and
   computers in Margaret Jacks Hall.  Ask me questions in
   simple English, and I will try to answer.  I am good with
   birthdays, phone numbers, room numbers, and the like.
   I am also very modest.
```

The last line of each "help" message comes from a random number generator.

```
# help
I know all sorts of things about the people, rooms, and
   computers in Margaret Jacks Hall.  Ask me questions in
   simple English, and I will try to answer.  I am good with
   birthdays, phone numbers, room numbers, and the like.
   If I can't figure out the answer, ask Joe Weening.

# quit

Good bye
End of SAIL execution
```

The sixth group, also programming in SAIL, consisted of DEF and JEH. Their program produced the following dialogue.

```
What is your question ??
Who is in room 306?

James Wilkinson is in room 306.
 What is your question ??
What is Ullman's phone number?

Douglas Baxter, Marsha Berger, James Boyce, Andrei Broder, James R.
Bunch, , Lew Creary, Elise De Doncker, Bob Floyd, and David Fouleer
are on the third floor.  Christina Fraley, David Fuchs, Dick Gabriel,
Anne Gardner, Jay Gischer, Chris Goad, Ron Graham, Daniel Greene, William
Gropp, and Mike Harrison.  John Herriot, John E. Hershberger, Jussi
Ketonen, Donald E. Knuth, Mark Lake, Fran Larson, Randall Leveque,
Ruth Andrea Levinson, Frank Liang, and Harry Mairson. Yonatan Malachi,
Jitendra Malik, Zohar Manna, Ernst Mayr,  John McCarthy, Jutta E. Mccormick,
Rosemary F. Napier, Stephen Nash, Joseph Oliger, and Palomar. Peter
Pepper. Prancing Pony, Vaughan Pratt, Robert Schreiber, Alan Siegel,
Tom Spencer, Alex Strong, Carolyn Talcott, Christopher Tong, and Luis
Trabb-pardo.  Lloyd Trefethen,  Jeff Ullman, Moshe Vardi, Paul Vianna,
Joe Weening, Richard Weyhrauch, James Wilkinson, Phyllis Winkler, Pierre
Wolper,  and Pat Worley.  Frank Yellin and Ignacio Zabala.
```

This looks like a list of everyone on the third floor. When some information, such as the phrase “are on the third floor,” has been printed, it is not printed again in the same answer. This explains the sentence structure above, as well as some of the answers to follow.

```
What is your question ??
Where is Gene Golub' ofice?

Gene Golub is in room 216 on the second floor.
```

The word “office” was mistyped; let's see if it makes a difference.

```
What is your question ??
Where is Gene Golub's office?

Gene Golub is in room 216 on the second floor.
 What is your question ??
What is his home address?

I don't have enough information to answer your question.
 What is your question ??
What is Gene's home address?

Gene is a PROFESSOR.
 What is your question ??
Where is the ladies' room?

I don't have enough information to answer your question.
 What is your question ??
Can I use a computer here?

Yes. DIRECTOR OF COMPUTER FACILITIES.
 What is your question ??
Where are the terminal rooms?

NA TERMINAL ROOM is in room 320 on the third floor. NORTH TERMINAL
ROOM is in room 460 on the fourth floor. SOUTH TERMINAL ROOM is in
room 408. TERMINAL ROOM is in room 343 on the third floor. is in
room 226 on the second floor.
What is your question ??
Where is the department secretary's office?

ASSOCIATE DEPARTMENT CHAIRMAN is in room 208 on the second floor.
SECRETARY is in rooms 368, 304, 340, and 326 on the third floor.
is in room 436 on the fourth floor. SECRETARY TO GENE GOLUB is in
room 214 on the second floor. ADMISSIONS SECRETARY is in room 206.
SECRETARY is in room 030 on the basement floor. DEPARTMENT SECRETARY
is in room 210 on the second floor.
 What is your question ??
Is there a xerox machine here that I can use?

Yes. XEROX COPIER and XEROX GRAPHICS PRINTER.
 What is your question ??
Where is the xerox on the fourth floor?
```

```
XEROX GRAPHICS PRINTER is in room 433 on the fourth floor.
 What is your question ??
What is Nancy' last name?

Nancy A. Dorio is a CSD RECEPTIONIST.
 What is your question ??
How many students are at extension 7-0926?

There are 8 such students.  Would you like to see them? (Y = yes) y
Richard J. Anderson, David Chelberg, David P. Helmbold, Sanjay Kasturia,
Timothy P. Mann,' Yoram 0. Moses, Ralph G. Saaveddra, and Andrew I.
Shore are PhD students with office phone 7-0926.
 What is your question ??
What are Dave's birthday and phone number?

Lia, Richard J., Paul, Andy, Daniel, Richard I., D. George, Bob, Len,
and Per are on the fourth floor.  Nelleke, , Rene, Avron, James, Denny,
Harold, Jake, Bruce, and William are on the second floor.  Dave, Harlyn,
Tom, Peter, David, Don, Mark, Martin, Ron, and Roy are on the basement
floor.  Douglas, Marsha, James, Andrei, James R., , Lew, Elise, Bob,
and David are on the third floor.  Kenneth P., Cindi L., James, David,
Kenneth, Jim, , Thomas E., Thomas, and Ginger H. are on the fourth
floor.  Paul, Claudia, Linda Gail, Nancy A., Bob, David A., Edward,
Jeff, Peter, and Mike are on the second floor.  Joan, Shel, Ross, Linda,
Erik, Ralph, David P., Lane, John D., and Ronald E. are on the fourth
floor.  Christina, Dick, Anne, Jay, Chris, Ron, Daniel, William, Mike,
and John are on the third floor.  Phil, Gene, Jose L., Lynn, Russell,
Milt, Susan, Dianne, Ingeborg M., and John are on the second floor.
John E., Jussi, Donald E., Mark, Fran, Randall, Ruth Andrea, Frank,
Harry, and Yonatan are on the third floor.  Jerrold, Anna R., Kevin,
Sanjay, Arthur, Michael, Scott, Guy, Gabriel M., and Curtis P. are
on the fourth floor.  Barbara R., Doug, Jock, Penny, Gordon, Thien-huong,
Mark, Glenn, Barbara, and Irmgild are on the second floor.  Amy, Howard,
Timothy P., Rick, Fumichiro, E.o., Jeffrey, Yoram O., Ben, and Mike
are on the fourth floor.  , Sidney, Michael R., Allan, Clayton Wai,
George, Marianne, Chris, and Rick are on the basement floor.  Jitendra,
Zohar, Ernst, Jutta E., Rosemary F., Stephen, Joseph, Peter, Prancing,
and Vaughan are on the third floor.  Bill, Peter, Mihoko, Harold, Oren,
Richard, Charles P., Larry, Jayne, and Stuart are on the fourth floor.
Charles, Neil, Ralph G., Art, Andrew I., Jorge, Steve, Marvin, Stephen,
and Gio.  Robert, Alan, Tom, Alex, Carolyn, Christopher, Luis, Lloyd,
Jeff, and Moshe are on the third floor. Betty, David E., Carolyn,
Allan, Marilynn, Bill, Terry, and Dawn are on the second floor.  Paul,
Joe, Richard, Phyllis, Pierre, Pat, and Ignacio are on the third floor.
Kai-zhi is on the fourth floor.
 What is your question ??
Who shares an office whith Dave Foulser?

David Foulser, Christina Fraley, Lloyd Trefethen, and Paul Vianna are
in room 316.
 What is your question ??
Who is Dave Foulser and what is his office?

David Foulser is in room 316.
 What is your question ??
Who is Dave Foulaer?
```

```
David Foulser is a PHD STUDENT.
 What is your question ??
Is Donald in room 328?

Yes.  Donald E. is in room 328.
```

And now, DEF tried an ambiguous question.

```
What is your question ??
Is Don in 326?

No; however, I can tell you this:  You seem to have specified contradictory
things.  Phyllis is in room 326. Don is in room 020G. Donald E. is
in room 328.  Gordon is in room 244.
 What is your question ??
What secretaries were born in November?

Jayne Pickering is a SECRETARY with a Nov. birth month. Marilynn
Walker is a DEPARTMENT SECRETARY.
 What is your question ??
Where are the altos?

ALTO is in rooms 241 and 226 on the second floor.   is in rooms 433,
460, 408, and 460 on the fourth floor.   is in room 320 on the third
floor.  ALTO WIZARD is in room 030C on the basement floor.
 What is your question ??
Which students in Lia's office do not have a home phone?

Lia Adams is a PhD student in room 416A with home phone 493-4962.
Nelleke Aiello is a staff member in room 224 with home phone 264-7609.
Almanor is a machine in room 241 with no home phone . Richard J. Anderson,
David Chelberg, David P. Helmbold, Sanjay Kasturia, and Timothy P.
Mann are PhD students in room 402. Dave Arnold is in room 022 with
home phone 962-0375.  Paul Asente is in room 416 with home phone 493-2013.
Rene Bach and Allan Terry are staff members in room 241 with no home
phone .  Harlyn Baker is in room 022 with home phone 966-9362.  Avron
Barr is a PhD student in room 246 with home phone 322-2233.  Douglas
Baxter is in room 313 with home phone 856-0728. Andy Bechtolsheim
      . . .
```

The program seemed to be printing out the entire database, so it was stopped here.

The last program to be demonstrated was by JLM and RJM. They were working in LISP. This program was built on top of an already-existing natural-language parser. Therefore, it would reject questions with syntax errors that other programs, looking only at keywords, might accept.

(A transcript of their output from the class was not available. Here are some sample interactions provided later by them. The questioner's input is shown on lines starting ‘Fire away: *’.)

```
Fire away: *HELP

Hi. I'm a program that will try to answer qustions about the people
```

```
and things in Margaret Jacks Hall.  Specifically, I know about names,
titles, rooms, phones, and birthdays.  Enter your requests in normal
English.  If we're both lucky, I'll understand and answer you.

Sample questions:

        Who is Lia?
        Which students in Lia's office do not have a home phone?
        Which professors are not on the third floor.
        Who is in the basement?
        How many professors are there?
        Where are the ALTOS?

Fire away: *WHO IS LIA
Ok...
Lia Adams, PhD student
```

'Ok...' means it parsed the question.

```
Fire away: *WHERE IS LIA
Lia, Room 416A.

Fire away: *WHO IS IN LIA'S OFFICE
Larry Paulson, PhD student, Room 416A.
Howard Larsen, PhD student, Room 416A.
Lane Hemachandra, PhD student, Room 416A.
Lia Adams, PhD student, Room 416A.

Fire away: *WHEN IS KNUTH'S BIRTHDAY
Ok...Oops.
Fire away: *WHAT IS KNUTH'S BIRTHDAY
Knuth, Birthday January 10.
```

The first question caused a problem because of a missing routine in the program.

```
Fire away: *HOW MANY PROFESSORS ARE THERE
Ok...
There are 21.

Fire away: *WHY
I'm sorry, I didn't understand that.

Fire away: *WHO IS A STUDENT
Ok...
The reply is 109 lines long. Print it anyway?
*N

Fire away: *which professors are not on the third floor
Ok...
Terry Winograd, associate professor, Room 236.
Gio Wiederhold, assistant professor, Room 438.
David Todd, visiting professor, Room 422.
Art Samuel, professor Emeritus, Room 436A.
Gordon Novak, visiting assistant professor, Room 244.
Doug Lenat, assistant professor, Room 230.
Gene Golub, professor, Room 216.
```

```
Mike Genesereth, acting assistant professor, Room 234.
Edward Feigenbaum, professor, Room 228.
Bruce Buchanan, Adjunct professor, Room 238.

Fire away: *HOW MANY STUDENTS DO NOT HAVE A HOME PHONE
Ok...
There are 20.

Fire away: *WHO HAS A BIRTHDAY IN JULY
Ok...
Richard Weyhrauch, research associate SAIL, Birthday July 3.
Moshe Vardi, visitor Ullman, Birthday July 4.
Carolyn Tajnai, department secretary, Birthday July 21.
Glenn Rennels, PhD student, Birthday July 12.
Ruth Andrea Levinaon, visitor of Trabb-Pardo, Birthday July 19.
Howard Larsen, PhD student, Birthday July 22.
Nancy A Dorio, CSD receptionist, Birthday July 3.
Elise DeDoncker, visiting professor, Birthday July 22.
Mark Crispin, systems programmr, Birthday July 19.
Douglas Baxter, PhD student, Birthday July 6.
Lia Adams, PhD student, Birthday July 24.

Fire away: *HOW MANY JIMS ARE THERE
Ok...
There are 2.

Fire away: *WHO IS JIM
Ok...
Jim McGrath, MS student.
Jim Davidson PhD student.
```

It would have been better to name the two Jims in response to the first question.

```
Fire away: *WHO IS AN ACTING PROFESSOR
Ok...
Mike Genesereth, acting assistant professor.
```

Solutions for problem 5

The solutions to this problem used a variety of approaches. In general, the problem was reduced to the subproblems of interpreting (parsing) the input questions, looking up the solution, and presenting the answer in natural language. Some programs also did preprocessing of the database, and handled input errors.

RJA/TPM classified input words as either words in the database, such as people’s names, or keywords, such as ‘how many’ and ‘birthday.’ Input words might match database entries even if they were slightly different: ‘December’ would match **‘Dec,’ and** ‘Computer’ would match ‘Alto.’ This would create an array of values or patterns to be searched for in each column of the database.

KPB/AH had a similar approach, using keywords in the input to indicate whether certain data fields were “filled” or “empty,” and using other keywords to recognize what type of question to answer (e.g., "for all entries which match the filled fields, print the empty fields," or "count the number of entries that match the filled fields and print the number").

ARK/JCM/AIS used multiple stages in their parser, which first created tokens from the input string, replaced certain ones by synonyms (to recognize plurals, for instance), and then matched them to information in the database, keywords such as 'who' indicating certain types of questions. Matching took place in three phases: exact, sub-word, and substring matches. There was even a provision for use of languages other than English.

JLM/RJM rejected the keyword approach, feeling that a useful program would need to be more sophisticated. Regular and context-free grammars were also deemed insufficient, and their program used a context-sensitive grammar, though most of the productions in that grammar were context free.

The database lookup procedure was quite strightforward in all of the solutions. It generally involved comparing each record in the database with the pattern generated by the parser, and deciding whether to pass information from that line to the output routine, or to count the number of occurrences of some type of match.

To create "user-friendly" output, JLM/RJM first tried to use a single "generic" template, designed to handle all forms of output from their program, but this was unsatisfactory. Their final solution used a set of templates, which depended on the fields to be output. The selection of the templates was done by the use of flags.

JEH/DEF also used templates, which were applied to sets of records being output, to translate the information in those records into natural-sounding English. This was repeated until all records had been printed.

RJA/TPM developed a set of heuristics for formulating output, which were applied in order until an answer was generated. These included several forms of error messages, indicating that the program didn't understand the question, or couldn't find the requested information.

The output of the program by ARK/JCM/AIS usually included some kind of restatement of the question. The reason for this was to clarify to the questioner any possible misinterpretation of the question. Various rules were applied to produce grammatically and syntactically correct responses, such as putting commas in the proper places in lists, removing trailing blanks, pluralizing when necessary, etc.

Most of the class found this problem to be the most enjoyable of the five.